PartnerWorld

Developers

IBM

Developer Connection

January 2000

Technical Magazine

PartnerWorld for Developers

e-business

e-business technology at your fingertips...

Welcome to the IBM* *Developer Connection Technical Magazine* – our offering to provide developers and business partners with key development information on exciting new technology areas such as XML, Linux, Java,• Enterprise JavaBeans and more! You'll find a wide range of articles written by experts–developers just like yourself–who want to share their expertise about the technologies you need to meet the complex demands of e-business.

1999 has been an exciting and challenging year and 2000 looks to be even more so. When you read this, we all should have a much better idea of what "y2k" really means! I know we all look forward to putting the "y2k" challenge behind us and getting on with the business of using new technologies to accelerate our companies' competitiveness in the world of e-business.

Our IBM Team is working to ensure that we provide you with the technical support, tools, and information you need and to lead in that e-business world. The main focus of our technical support team is you–the developer. Our mission is to put the resources you need at your fingertips to help you take advantage of IBM technology and integrate the power of e-business in your solutions. Developer Connection and Developer Support Online are key components of our technical support strategy for supporting developers.

The Developer Connection delivers e-business tools and software products that can be downloaded from the Developer Connection Web site at www.developer.ibm.com/devcon/. This information also is available on CD. Developer Support Online, at www.developer.ibm.com/welcome/technical1.html provides the answers to technical questions that can help you accelerate your development efforts. Our latest deliverable is a set of Integration starter kits for e-business, downloadable packages that show you how to implement e-business solutions for specific technologies.

One of the most important ways we interact with you is at IBM conferences that focus on business partners and the developer community. You can expect to see us at PartnerWorld 2000, formerly the IBM Business Partner Executive Conference (BPEC), January 23-26, in San Diego, California.

The theme for PartnerWorld 2000 is "The Real Business of e-business." Conference highlights include operational, scaleable, robust, and secure e-business systems with a special track of forums, seminars, demonstrations and activities designed specifically for your needs.

Be sure to stop by the PartnerWorld for Developers Technical Support booth at the conference to find out about the latest tools available from the Developer Connection and see a demo of the Developer Connection and Developer Support Online on the Web.

I look forward to meeting and talking with you at PartnerWorld 2000 and at other events throughout the year. Give us your feedback, and let us know how we can help you be competitive in the world of e-business!

Roy Abo

Manager,
Solution Developer Marketing Technical Support

In this issue

XML: A "Way Cool" way to enhance your business applications

by LindaMay Patterson and Rick Stevens

XML, an acronym for eXtensible Markup Language, is a hot new e-business technology. *So what is XML and what does it mean?* XML provides a standard for creating specific markup languages used to describe information. Being extensible means that there is no predefined "limited" tag set (like HTML). Each industry or business environment can define a tag set that fits its particular needs. A pretty powerful concept.

Our objective with this article is not to explain the basics of XML. Our objective is to give you a sense of the value of XML from a business and technical perspective and to delve into four key areas where we feel XML is going to help reshape the way business applications operate in the very near future. The underlying premise for that bold statement comes from our assertion: Once business data is placed in a self describing data model (via XML) that data can be exploited in ways unimagined with current technology. For example, the same information, described in XML format, could be used to communicate with a hand-held device, support sophisticated search methods on your Web site and be used for data interchange with other applications across the Internet.

XML adds value

Often when we (technologists) talk about new technology, we get caught up in the purely technological reasons for using the technology. We must always remember there has to be a customer (business) that can use the technology to solve specific problems or leverage new opportunities. That is why we have broken the value of XML into two parts: the business value and the technical value.

Business value

To provide true business value, a technology must make doing business easier and more efficient or provide a means to help increase market share. XML assists in both scenarios. First, it provides business information in the form of XML documents to an increasingly mobile work force and an extremely varied customer. Furthermore, once information is encoded in XML, it can be viewed and manipulated by a wide range of devices, ranging from cell phones and other hand-held devices driven by Wireless Markup Language (WML); to desktop computers with browsers that render HTML (XML can be transformed into HTML); and by the latest browsers that work with XML directly. In each case, XML serves as an abstract representation of the information destined for user consumption that can be easily transformed to and from a device-specific format.

Secondly, XML allows business data contained within databases to be packaged into a highly portable format for use by a wide number of interested parties. For example, electronic catalogs that are created from product data stored in company databases can be represented in XML and viewable on the Internet. When represented as XML, these catalogs become more easily accessible by search engines and agents, which assist consumers and companies looking for the right product at the right price. The XML data can be queried to extract information on products of interest. Also, XML provides a consistent representation of data that can be used by disparate applications. Mergers and acquisitions often force companies to integrate applications used in various environments. By using XML as the baseline for information, these applications have a common communication vehicle.

Once business data is placed in a self describing data model (via XML) that data can be exploited in ways unimagined with current technology.

Electronic data interchange (EDI) has a history of allowing businesses to communicate with their partners to improve their efficiency and effectiveness. However, EDI has a relatively high infrastructure cost and requires knowledge and use of sometimes cryptic data formats. XML documents are self describing, exchangeable over the Internet and easily consumed by the receiver. Industries that have not used EDI are looking to XML as a viable alternative.

All these reasons add up to providing better communication with customers, business partners and within the company itself, which can have a direct impact on the company's effectiveness within the marketplace.

Technical value

There are purely "being high tech" reasons for using a new technology. However, in some cases the long term benefits of the investment may be questionable. XML provides many opportunities to enhance an application or support new development while providing immediate benefit. XML provides a non-invasive way to make existing applications "e-business," by supporting extraction of existing data into a form that is usable on the Web. XML accomplishes this by becoming the abstraction layer that hides the physical representation of information that may be generated from multiple data sources and manipulated by multiple business processes. In the near future, XML will likely become a popular choice for describing data maintained in database and file systems.

Internationalization has always complicated the ability of business applications to be used in various countries. IBM Rochester minimized that problem for the AS/400* by allowing the definition of the user interface in an XML implementation known as Panel

Definition Markup Language (PDML). PDML allows abstract definition of the user interface (UI) layouts, making it easier for National Language Support Teams to translate the text definitions to other languages without requiring knowledge of the user interface implementation or target language used (Java, C++, ...).

Technical value is making technology work to provide a meaningful business solution. Each technology should be selected based on its ability to bring value to a business. In this regard, XML brings a lot to the table.

XML potential

Although XML is shaping up to be a pervasive technology with application in a broad number of areas, we see four scenarios where XML is emerging as a very popular technology choice. This does not mean that XML only plays in these areas, but XML plays well in these areas and has been used to develop some very innovative and effective solutions. These four areas are:

- Integrating thin client and pervasive computing devices with business applications
- Simplifying Electronic Data Interchange capabilities
- Encouraging application integration
- Enhancing the utility of Web content

Each of these areas can have an immediate positive impact on the way a company operates. ***Figure 1*** depicts a business environment taking advantage of XML in the various ways we have identified.

Thin client/pervasive computing devices

The combination of a mobile work force and the emergence of a variety of personal devices is making an impact on businesses and business applications. Times are changing so fast that application providers need a means to make in-house information readily available to these mobile workers and customers. Solutions that are very client-centric are very difficult to maintain given the mobility of the work force. Trying to evolve existing user interfaces to this environment can take time and extensive resources. One mechanism available to enable thin-client

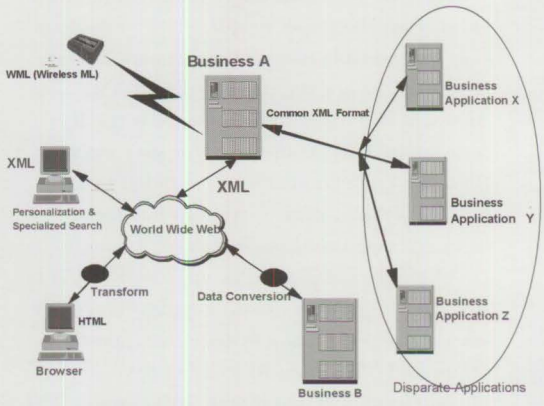


Figure 1. Taking advantage of XML

and mobile-client access is a concept known as transcoding. Transcoding takes an existing data stream (for example a 5250 or 3270 data stream) and converts it into XML format. The XML format then can be sent to any device capable of accepting and displaying XML data, or the XML data is converted (transformed) to a format that is more appropriate for the target device. In many cases, style sheets are used to

CONTINUED ON PAGE 4

Putting the pieces together for your e-business

TAKE ADVANTAGE OF THE LATEST OFFERING FROM THE IBM DEVELOPER SUPPORT — INTEGRATION STARTER KITS FOR E-BUSINESS. THE KITS CAN HELP YOU IMPLEMENT YOUR E-BUSINESS SOLUTION QUICKLY AND EASILY. INTEGRATION STARTER KITS FOR E-BUSINESS ARE DOWNLOADABLE PACKAGES OF TECHNICAL INFORMATION THAT OFFER INSIGHT INTO TECHNICAL PROBLEMS THAT DEVELOPERS MAY ENCOUNTER AND INCLUDE A COMBINATION OF WORKING CODE, TECHNICAL WHITE PAPERS, FREQUENTLY ASKED QUESTIONS (FAQS), PRESENTATIONS AND LIVE DEMOS.

CURRENTLY, THERE ARE SIX INTEGRATION STARTER KITS AVAILABLE:

- CONVERTING XML TO HTML USING XSL: DEMONSTRATES HOW TO CONVERT AN XML DOCUMENT TO HTML FORMAT WITHOUT PROGRAMMING USING TOOLS AVAILABLE FROM THE IBM ALPHAWORKS WEB SITE.
- ENTERPRISE JAVABEANS (EJB): DETAILED ILLUSTRATIONS, SAMPLE CODE AND INSTRUCTIONS ARE PROVIDED TO LEAD DEVELOPERS THROUGH THE PROCESS OF CREATING, TESTING, AND DEBUGGING AN ENTERPRISE JAVABEAN (EJB) WITH CONTAINER-MANAGED PERSISTENCE (CMP) USING VISUALAGE* FOR JAVA, ENTERPRISE EDITION AND EXISTING DATABASE TABLES IN UNIVERSAL DB2.* ALSO SHOWS HOW TO DEPLOY THE EJB IN IBM WEBSPHERE* APPLICATION SERVER (ADVANCED) AND COMPONENT BROKER ENVIRONMENTS.
- IBM SANFRANCISCO/SERVLET: LEADS DEVELOPERS THROUGH THE PROCESS OF CREATING AND TESTING A THIN-CLIENT GATEWAY (JAVA SERVLET) TO A LOGICAL SANFRANCISCO NETWORK.
- NET.COMMERCE AND MQSERIES*: INSTRUCTIONS AND A SAMPLE JAVA PROGRAM DEMONSTRATE HOW A CUSTOMER ORDER PLACED ON A NET.COMMERCE MANAGED ONLINE STORE IS ROUTED TO A BACK-END SYSTEM, VIA MQSERIES, AND PROCESSED BY A SAMPLE JAVA PROGRAM.
- ONLINE BANKING (JAVA SERVER PAGES): ILLUSTRATES A WEB APPLICATION MODEL BASED ON THE USE OF JAVA SERVER PAGES (JSPS) AND COMMANDS.
- ONLINE CALENDAR DEMO: ILLUSTRATES A DYNAMIC, INTERACTIVE WEB APPLICATION BUILT ON IBM WEBSPHERE APPLICATION SERVER AND IBM UNIVERSAL DB2.

VISIT THE PARTNERWORLD FOR DEVELOPERS TECHNICAL SUPPORT WEB PAGE AT WWW.DEVELOPER.IBM.COM/TECH/INTEGRATION/INDEX.HTML FOR ADDITIONAL INFORMATION OR TO DOWNLOAD INTEGRATION STARTER KITS FOR E-BUSINESS.

CONTINUED FROM PAGE 3

manipulate the XML data into an optimum format for the target device.

The possible uses of this concept are far reaching, leveraging the benefits of XML data that can be reused in a variety of ways. By moving to XML as the basis for user interaction, an application can be extended to support new client devices without having to undergo significant modification.

A good example of this approach would be the extension of an existing application, such as a travel reservation system, to support generation of XML from its existing device-specific data format. When it is in XML format, XSL stylesheets could be applied to the data to convert it to Wireless Markup Language (WML), which can be used to drive interfaces on various wireless devices such as cellular telephones. Such an approach would add value to the original application by supporting query and update of travel itineraries from handheld, mobile devices.

Electronic Data Interchange (EDI)

Various industries have a large investment in EDI. EDI has worked well for relationships between medium and large companies that have the resources to handle and process EDI transmissions. However, in order to deploy an EDI solution, expensive Value Added Networks (VANs) must be established to transport the information. There are a variety of industries that have not gotten on board with EDI and smaller companies that could not handle the expense of doing EDI. These industries and companies are getting into the interchange game via XML. XML data can be exchanged over the Internet, thereby eliminating the need for the VAN infrastructure. A group of businesses within an industry can determine a mutually acceptable tag set and start sharing information quickly and easily using XML. Because XML has an unlimited tag set, new tags can be added and shared while minimizing application impact.

EDI has focused on the high volume and repetitive transaction communications between businesses. XML can be used to handle less frequent communications and can drive both an immediate increase in business process automation and the sharing of information between businesses. As additional protocols are developed around XML, XML will become an even more important player in the EDI space.

Application integration

In the value section of this article we touched on the need for disparate applications and business systems to communicate with each other. XML provides an intersection point for all these diverse applications. For example, various organizations within an insurance company need to view and handle an auto insurance policy. The organizations range from initial policy writing through policy evaluation and approval and, of course, the policy is essential to the claims adjusters (any of these players could be an outside group). Often the systems that support these areas were built specifically to fit the users' needs or sometimes the parent company acquires other businesses that specialize in an aspect of the overall business process. By representing the policy and policy history as XML document(s), these varied systems and users have a common representation of the information to work with. Using this approach ensures the information is available for different situations; it can be shared on the Web to interested parties or it can be used by these disparate business applications. The filtering ability supported by style sheets can be used to tailor information presented based on the needs of the audience.

Enhance Web content

HTML makes all types of information available via the Internet. HTML focuses on the presentation of data but does not preserve the meaning of the data. XML documents consist of self describing data, which allows the data to retain its intelligence, rather than becoming purely text. Searching for information on the Web is very imprecise today, given the volume of information and the lack of support for defining a precise context for the information of interest. When data is in an HTML format, there is no way to restrict a search. For example when looking for people whose names are "chip;" you would get hits on potato chips, chocolate chips, wood chips and other potential uses of the term "chip," in addition to what you were actually looking for. When data is in XML format, you can specify that you want to find all persons whose last name is "Chip", thus avoiding other contexts in which "chip" is used. Because XML focuses on representing the data and does not determine the data presentation, the reuse and versatility of an XML document is increased. Other technologies like XSL (extended Stylesheet Language) provide the ability to create different views that allow tailored and customized presentation, based on user preference.

In conclusion

The areas discussed here represent possibilities that are fast becoming reality. The mechanisms needed to make these scenarios happen are surfacing in the market. Opportunities abound to leverage XML to provide immediate value to business, to solve existing problems and to leverage new opportunities. XML is becoming the universal means of representing information. Given the important role that information plays in today's world, it's no wonder that XML is used in so many different situations to solve so many problems. The opportunities to leverage XML are limitless, only constrained by our creative abilities and our aptitude for innovation.

LindaMay Patterson *is an Advisory Software Engineer in Partners in Development at IBM Rochester. She has worked at IBM for almost 25 years in various Business Application environments. Currently, she is part of the Partners in Development Java team working with Enterprise JavaBeans and is the area XML Focal Point. Prior to this, she worked on the IBM SanFrancisco product, developing the education packages, and in England and Germany helped to define the SanFrancisco's product content. Her application background is primarily in Distribution and Logistics Systems. LindaMay can be reached at lindamay@us.ibm.com.*

Rick Stevens *is the Chief Engineering Manager for application development technologies on AS/400 at IBM Rochester. He is responsible for AS/400's strategy to support XML, including definition of plans to provide tools and other facilities required by applications to exploit the use of XML on AS/400. Rick can be reached at rstevens@us.ibm.com.*

XML: How it will change the Web

by Doug Tidwell

The Extensible Markup Language (XML) is currently being hyped as a technological achievement similar in scope to sliced bread and the wheel. Beneath the hoopla, XML represents the next evolution in Web technology. In this article, I explore the basics of XML and discuss how it will change the Web.

Why do we need XML?

XML is a new technology for Web applications. XML is an official recommendation of the World Wide Web Consortium (W3C) that lets you create your own tags. Its heritage comes from the Standard Generalized Markup Language (SGML), a markup standard created by Dr. Charles Goldfarb.

When people first hear about XML, some common questions are:

- "Why do we need another markup language?"
- "Everybody's browser supports HTML today, so why create more tags?"
- "Given that lots of HTML tags haven't been implemented the same way by the big browser vendors, why let anybody and everybody create their own tags?"

The answer is...

The answer to these questions is that HTML and XML serve different functions: HTML tags describe how things should be rendered on the screen, while XML tags describe what things are. Put another way, HTML tags are designed for the interaction between humans and computers; XML tags are designed for the interaction between two computers.

To illustrate this difference, let's look at the HTML and XML versions of a short document. Here's the HTML version:

```
<p><b>Mrs. Mary McGoon</b>
<br>
1401 Main Street
<br>
Anytown, NC 34829</p>
```

When this document is rendered in a browser, it looks something like this:

Mrs. Mary McGoon
1401 Main Street
Anytown, NC 34829

Anyone familiar with postal addresses in the United States will recognize this document as someone's address. Even if you're from another country where postal codes and other conventions are different, you can still surmise that this is someone's address. Imagine writing code to interpret this document, however. To extract the zip code from this address, our algorithm might look like this:

```
Given a <p> tag that contains two <br> tags, take the
text of the second <br> tag. In that text, everything up to
the comma is the name of the city, the two-character
token following the comma is the name of the state, and
the final token is the zip code.
```

While this algorithm would work for our sample HTML document, it's easy to think of a perfectly valid address that breaks our algorithm. We've also completely sidestepped the issue of distinguishing a <p> tag that contains an address from any other <p> tag. While the address formats beautifully in a browser, our HTML markup isn't nearly as well suited for use by a program.

An XML version of the same document is shown in ***Listing 1***.

As with our HTML document, anyone familiar with U.S. postal addresses will recognize this

XML will change the Web because of its power and flexibility as a data interchange format.

```
<?xml version="1.0"?>
<address>
 <name>
  <title>Mrs.</title>
  <first-name>Mary</first-name>
  <last-name>McGoon</last-name>
 </name>
 <street>1401 Main Street</street>
 <city>Anytown</city>
 <state>NC</state>
 <zipcode>34829</zipcode>
</address>
```

Listing 1.

document as an address. More importantly, a computer can recognize the parts of this address as well. Here's a much more robust algorithm for finding the zip code in our XML document:

```
The zip code is the text of the <zipcode> tag.
```

Obviously this code is much simpler to write, and it's difficult, if not impossible, to write a valid address that breaks this algorithm. A computer can understand all of the parts of the address and how they relate to each other, and can decide the best way to render that data. For example, our XML document might be rendered like this:

Mrs. Mary McGoon
1401 Main Street
Anytown, NC 34829

In rendering the XML tags in this style, we might convert them into HTML markup that's virtually identical to our earlier HTML document. If we wanted to print a mailing label for this address, we might render our document like this:

5-Digit Presort 34829
McGoon, Mrs. Mary
1401 Main Street
Anytown, NC 34829

In this case, we've printed Mrs. McGoon's zip code as a bar code for the benefit of the scanners at the post office. The most important concept here is that content and presentation are separate. The data and its structure are tagged in a presentation-independent way, and the decision of how to render it is delayed as long as possible.

CONTINUED ON PAGE 6

CONTINUED FROM PAGE 5

Before moving on to discuss how XML will change the Web, a final point: XML will not replace HTML. As stated earlier, the two markup languages are designed for different purposes, and both will coexist on the Web.

How XML will change the Web

Now that we've discussed how XML works, how will it make an impact on the Web? There are several things that likely will happen as XML technology is established.

Enable universal data

Looking at the Web today, you'll find several universal technologies: TCP/IP, HTML, and Java. TCP/IP is the universal connectivity protocol; everything from mainframes to laptops to cellular phones can connect to the Web using it. HTML is the universal rendering language. Although not all browsers support all functions, there is a core set of HTML tags that can be rendered on any browser. Finally, Java's promise of "write once, run anywhere" makes supporting the wide variety of devices on the Web much easier.

Because of these ubiquitous technologies, it's relatively straightforward to create a Web application that runs on any platform. XML completes the picture by enabling universal data. I can build an XML document that describes a data structure, and that structured data can be sent anywhere across the Web. XML should change the Web because of its power and flexibility as a data interchange format.

Enable business-to-business communication

One of the challenges in conducting e-business is communicating with other organizations, whether they are partners, suppliers, competitors, or even other parts of the same company. XML greatly simplifies business-to-business communication because the only thing that any two organizations have to agree upon is the XML tag set that is used to represent data. Neither organization has to know how the other's back-end systems are organized. If my systems run OS/390* and your systems run Linux, that doesn't matter. If my databases are relational and yours are object-oriented, that doesn't matter. If my code was written in C++ and yours was written in Java, that doesn't matter. The only thing that is important is that we agree on a standard set of tags for data interchange.

Once we've agreed on a tag set, each of us can write the mapping code to transform XML documents into whatever format we need to work with our back-end systems. For example, an XML document that's received from a partner might be parsed, then converted into a transaction that drives some business process on my system. Even better, if another company joins our consortium, we don't have to write more code to interact with the systems of the new company. We simply require that company to follow the document rules we defined in our XML tag set.

Enable smart agents

When writing an agent, one of the challenges is to make sense of incoming data. A good agent interprets information intelligently, then responds to it accordingly. If the data sent to an agent is structured with XML, it's easier for the agent to understand exactly what the data means and how it relates to other pieces of data it may already know. As we illustrated in our sample HTML document, writing code to interpret the data contained in HTML tags is difficult and error-prone. With XML, the structure of the data is easily determined and manipulated.

Enable smart searches

One major problem with today's Web is that search engines can't process HTML intelligently. For example, if you're looking for someone named Chip, you might get pages on chocolate chip cookies, RAM chips, poker chips, and guys named Chip. On the other hand, if you were searching for documents that contained a <first-name> tag with a value of "Chip," you would get much better results. Being able to constrain searches to those XML documents that use a certain set of tags would allow you to weed out a massive amount of unrelated content.

As an aside, being able to constrain search results to documents that use a particular tag set is one of the market forces that should drive the acceptance of XML. Say that a group of automobile dealers defines a tag set for used cars, and that several popular search engines promise great results because their search engines look only at XML documents that use those tags. If you're an auto dealer, you can either join the market and support that tag set, or you may be left out of the market completely. If your stock isn't described using the standard XML markup, any would-be car buyers using an XML search engine would not find you.

Summary

XML is poised to change the Web, enabling a whole new generation of e-business applications. In this article, we've reviewed the basics of XML, shown how it complements HTML and other Web technologies, and discussed several ways in which XML will have an impact. Just as HTML and graphical browsers sparked an exponential growth in Web use, XML's enabling of e-business will start another period of exponential growth. Let's get started!

Resources

- The XML Specification is available at www.w3.org/TR/REC-xml.
- The developerWorks home page is at www.ibm.com/developerWorks.
- The "Introduction to XML" tutorial is available from the XML zone on developerWorks at www.ibm.com/developer/xml.

Doug Tidwell *is a Senior Programmer at IBM. He has been working with XML-like applications for several years and he helps IBM customers evaluate and implement XML technology. He holds a master's degree in Computer Science from Vanderbilt University and a bachelor's degree in English from the University of Georgia. You can contact him at didwell@us.ibm.com*

IBM software for Linux

by Peyen Fong

The recent rapid growth of Linux as a server platform is creating significant demand for applications. Analysts estimate that 17% of Intel server deployments (750,000 servers) in 1998 were on the Linux platform. Estimates of the installed base in 1998 put Linux at more than 10 million users. According to surveys, 58% of all Web servers run open-source operating systems such as Linux and FreeBSD.

In a recent survey of Value Added Resellers who attended a Caldera Systems Tour in North America, 62% expected to deliver a Linux solution within six months, and 82% expected to deliver one within 12 months. They are keenly interested in software that can help them deliver Internet, intranet, database, mail, Web and e-commerce servers. As many as 15% of all developers using UNIX report using some Linux, as well. A survey of 600 IT managers found that the leading application that they expect to deploy on Linux within the next 12 months is IBM WebSphere. Two-thirds of those surveyed said that they were incrementally adding Linux to their environments.

A large portion of the deployments to date have been Web servers and file and print servers. A significant part of the next wave of growth on Linux will include Web applications, local and departmental business applications and applications that integrate Linux with host and other systems that are part of a customer's heterogeneous environment. To help developers and customers take advantage of this next wave of growth, IBM plans to provide the database, collaboration, host connectivity and file-sharing applications.

What's in it for Solution Developers?

IBM assists developers who are developing Linux applications by offering hardware solutions, powerful and secure software and technical support that even includes support for the Linux operating system, itself. With one call, developers can get help while building applications. Solution developers also can leverage the Netfinity Server Proven Program, which recently announced support for Linux. Developers can visit IBM's worldwide porting centers for porting assistance and to certify their applications on Netfinity. For more information about the Netfinity's Server Proven Program see www.pc.ibm.com/us/netfinity/serverproven. To help developers build skills in specific development and support areas, IBM Education and Training announced a comprehensive curriculum of courses on Linux at IBM's Solutions '99 developer conference this past July. For more information about IBM's educational offerings on the Linux operating system, visit the Web site at www.ibm.com/services/learning/linux.html.

IBM software for Linux in 1999

IBM is providing software that makes it easier to develop applications that can leverage this next wave of growth. IBM has delivered or plans to deliver the following on Linux in 1999. Many of these products are available for download from the Developer Connection online catalog at the Premier level.

- **DB2 Universal Database Version 6.1,** for Linux shipped in July. DB2 Universal Database is a multimedia, Web-ready relational database management system. Version 6.1, DB2 Universal Database for Linux provides significant enhancements to ease the development of e-business applications. The DB2 Personal Developer's Edition is available at www.software.ibm.com/data/db2/linux.
- **IBM WebSphere Standard Edition** shipped in the fall of 1999. IBM WebSphere is a set of software products that help customers develop and manage high performance Web sites to ease the transition from simple Web publishing to advanced e-business Web applications.
- **Lotus® Domino® for Linux** was announced at LinuxWorldExpo in August and a sneak preview version is available for free download from www.notes.net/linux. The product was released in December 1999. The Domino Server Family is an integrated messaging and Web application software platform for growing companies that need to improve customer responsiveness and streamline business processes.
- **MQSeries Server for Linux,** part of the IBM MQSeries Family provides an open, scalable, industrial strength messaging and information infrastructure, enabling enterprises and beyond to integrate business processes.
- **Host on-Demand,** which shipped in March 1999, is a terminal emulator for Linux. Get secure browser access to host information, Java-based emulation, and e-business application programming support with IBM eNetwork Host On-Demand Version 3, a SecureWay Software offering. Access a number of TN3270E, TN5250, VT50/100/220 and CICS sessions and still use the browser for other tasks. Host On-Demand offers standard desktop utilities, file transfer, host print, a graphical user interface (GUI), and Java programming tools. It requires no client installation or middle-tier server.
- **Advanced File System** (AFS) from Transarc shipped in December of 1998. AFS Enterprise File System from IBM makes collaborative work possible by allowing authorized users to share files–across the hall or around the world. AFS server and client for Linux offer a cost-effective, reliable and scalable file sharing option. They enable interoperability between and among servers and clients for Windows NT and

The next wave of growth on Linux will include Web applications, local and departmental business applications that integrate Linux with host and other systems.

CONTINUED ON PAGE 8

CONTINUED FROM PAGE 7

UNIX operating systems.

- The **IBM Developer Kit for Linux, Java Technology Edition** is available on the Web and a version of VisualAge for Java for Linux is being developed. Visit the IBM Web site at www.ibm.com/linux often for additional information about these products.

Partnering with the Linux community

IBM is working hard to 'do Linux right' and to support the Linux development community through the following avenues:

- Active memberships in Linux advocacy and standards groups, such as Linux Standard Base (www.linuxbase.org) and Linux International (www.li.org).
- Linux tradeshows (such as LinuxWorldExpo, and Linux Expo)
- Contributions of open source code (such as Jikes Java Compiler (www.alphaworks.com) and Secure Mailer software).
- Support, education and marketing of Linux solutions with four commercial vendors of the Linux operating system: Red Hat, Turbolinux, SuSE and Caldera Systems.

For more information about IBM and Linux visit the Web site www.ibm.com/linux.

Peyen Fong *is a Senior Marketing Manager for Linux Software in the IBM Software Group. Her responsibilities include worldwide market management and planning for IBM software on Linux. Peyen received her MBA in Marketing and International Business from New York University Stern School of Business. She holds a BA in Economics and Asian Studies from Cornell University.*

IBM sets another disk-drive world record

In October, IBM announced that it had set a new computer data storage world-record of 35.3 billion data bits per square inch on a magnetic hard disk — a 75 percent increase over the 20-billion-bit milestone the company achieved less than five months ago.

This new record is expected to lead to disk drives that could store three times more information than those available today.

The most significant feature in IBM's latest achievement is the new magnetic "media" — the metal-alloy materials that coat the hard-disk platters and where the data is stored. Particularly important, this proprietary magnetic material exhibits product-quality stability, not the data-robbing fluctuations that had been feared to be present at such high densities.

Data is written onto the media as a pattern of bits — tiny oblong regions magnetized in either of two opposite directions. If bits can be made smaller, more data can be stored within the same disk area. But if the bits become too small, they may not be able to maintain their magnetic orientations for the many years required for commercial products.

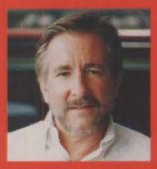
One of several ways to forestall this "superparamagnetic effect" is to use a magnetic material that more strongly resists magnetization changes. But, paradoxically, it also must still be easy to erase and rewrite when needed. IBM's new magnetic media has both of these desirable properties. The test bits were as stable as those in products today. The proprietary disk media can be manufactured commercially using existing production equipment. Lab results also suggest that even smaller bits on this media will continue to be stable, enabling a clear path to even higher area densities in the future.

At 35-gigabit density, every square inch of disk space could hold 4.375 gigabytes — nearly as much data as a 4.7-inch (120mm) diameter DVD-ROM. (4.7 gigabytes per surface) or seven CD-ROMs (each 650 megabytes). At this record density, a single desktop drive platter (3.5-inch diameter) would hold nearly 50 gigabytes; a notebook platter (2.5-inch) more than 20 gigabytes and a microdrive (1-inch) more than 2 gigabytes.

Because fewer disks are needed to achieve a given data-storage capacity, increasing data density leads to disk drives that are lighter and consume less energy — important factors in portable computers — and tend to be more reliable.

For more information about IBM hard-disk drives, visit the IBM Storage Systems Division Web site at www.ibm.com/harddrive

An object design for a *Java RMI Native* Program Server



by Keith Wright

Editor's Note: This article is a shortened version of a white paper detailing a proven object design for a medium-sized Java RMI native code server. The white paper and accompanying sample code are available on the Developer Connection Web site at www.developer.ibm.com/devcon/.

This article presents a proven design for a particular kind of Java Server: one that allows one or more remote network users to start and communicate with a batch of native code that accesses files and a database. This design is suitable for commercial departmental-level servers (less than 50 simultaneous users) with a modest amount of data to be passed between the client and the server through remote method arguments. The clients are not browser-based, but all machines are assumed to be running a Java 1.1.7 Virtual Machine (JVM).

To successfully run the example code, you would need a specific DB2 database. However, you can compile the code and run it until it throws an SQL exception.

Before we begin, it is important to keep in mind the following requirements that guided our Server design.

- Client and Server must be able to communicate with each other via shared database tables and files, as well as via remote method arguments.
- Each of several clients must be able to simultaneously call methods on the Server to do the following:
  - Register (and unregister) itself as a client of the Server.
  - Start batch jobs on the Server.
  - Restart a failed job.
  - Get the status of all jobs.
  - Cancel any job at any time.
- It is important to keep in mind that the server must be able to call back, that is, call methods on the client to do the following:
  - Report a job's status.
  - Display an error message to the user.

Class and thread design

Figure 1 shows the most important Java classes in our design. As you can see, a ClientApplication object sends a JobRequest object to the ServerApplication object. This is accomplished by a Remote Method Invocation (RMI) call. From the JobRequest object, the ServerApplication object creates Job objects and pushes them on the JobQueue object, where they remain until their turn to run. Only one job is allowed to run at a time. First come, first serve.

Each job consists of two phases; one is a native process and the other is a thread in the Server's JVM. When a job is running, it reports its status to the client by remote methods in the client's JVM. Further communication between the ServerApplication and ClientApplication objects can occur through shared files and database tables.

Most of the difficult threading is taken care of automatically by RMI: RMI ensures there are threads available to handle many simultaneous remote method calls from clients. RMI is a pure-Java answer to remote procedure calls (RPC), the Distributed Component Object Model (DCOM), and the Common Object Request Broker Architecture (CORBA). The RMI model aims to make designing distributed applications as simple as designing non-distributed applications.

RMI ensures there are threads available to handle many simultaneous remote method calls from clients.

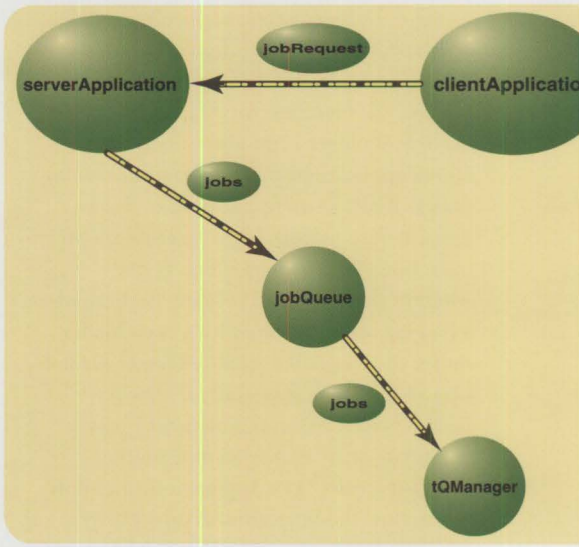


Figure 1. Object diagram

In addition to the Java supplied threads, we introduced two additional threads: Tjob and TqManager. The TqManager object is responsible for listening to the job queue and starting a job when appropriate. In our design, only one job at a time is allowed to run: the TqManager thread is blocked when another job is running or there are no jobs in the job queue. In other words, TqManager wakes up only when a job is ready to start, and then only long enough to start its Tjob thread object. Then it goes back to sleep. Hence, TqManager is blocked most of the time.

The most important class in our design is ServerApplication.

Design of Class ServerApplication

The ServerApplication class extends UnicastRemoteObject and implements the interface IR1serverApplication. (Our naming convention specifies that variable names that begin with I denote interfaces. If they are followed by the letter R, then they denote remote interfaces. The numeral following the R provides a unique identifier for the interface.) In JDK 1.1.7, UnicastRemoteObject is the only suppported implementation of the RemoteServer interface and, as such, is the only defined way to create remote server objects. But non-subclasses of UnicastRemoteObject can be exported if they implement a RemoteServer interface and

CONTINUED ON PAGE 10

CONTINUED FROM PAGE 9

export themselves using the exportObject () method of the UnicastRemoteObject class.

Following are the major processing steps in class ServerApplication:

- Create a new instance of ServerApplication.
- Set the RMI Security Manager.

  The RMISecurityManager class defines a default security policy for RMI applications (not applets). RMISecurityManager extends the class SecurityManager. SecurityManager is an abstract class that allows applications to implement a security policy. It allows an application to determine, before performing a possibly unsafe operation, what the operation is and whether the operation is being performed by a class that is created by a class loader rather than installed locally. Classes loaded by a class loader (especially if they have been downloaded over a network) may be less trustworthy than classes from files installed locally. The application can allow or disallow the operation. If no security manager has been set, RMI will load classes from only local system files as defined by CLASSPATH.
- Authenticate the user who starts the server JVM.
- Initialize log files.

  The log system is the principal means by which the server is supported in the field. Log messages should be clear, regular and easily located. The log is partly a file and partly a database table. For national language translation support, log messages originate from a disk file managed by the Message class.
- Read and validate the properties file.

  Java provides the Properties class as a convenience for reading configurable parameters from disk. As you can see in the code, the Server's parameters include its Internet Protocol (IP) name, port number, name of the log file, URL of the DB2 database server and the qualifier for the DB2 database tables. (This qualifier is a DB2 instance name.)
- Connect to the database.
- Initialize the job queue and queue manager.
- Get the resource bundle.

  Getting the resource bundle and messages is done in the Message class constructor. In this example, the resource bundle is actually a properties file. This file is used to make it easy to support a variety of national languages. There are no hard-coded message strings in the Server. Therefore, a separate disk file for each language supported can be packaged for a particular country. For example, if your server is sold in Germany, you would package a German message file. For English speaking countries, you would package an English message file.
- Load the translated Message strings.

  The server loads all the messages into memory when the Server starts. This is to detect an incomplete message file before the server begins accepting job requests. Otherwise, client jobs might fail because of an incomplete server message file. The penalty for this is the memory needed to hold all the messages.
- Start the RMI registry server process.
- Put the Server's name and object in the RMI Registry.
- Start the job queue main process.

The preceeding discussion summarizes the highlights of how a server starts and controls jobs. The next summary discusses the server from the viewpoint of the client.

Summary of the IR1ServerApp Interface

The following six methods, declared in interface IR1Serverapp, are available on the Server for any client to call remotely via RMI.

1. public void addClient(IR1clientApplication aClientObj) throws RemoteException, Exception.
2. public void removeClient(IR1clientApplication aClient Obj) throws RemoteException, Exception.
3. public JobID createJob(JobRequest aJobRequest) throws RemoteException, Exception.

   The immediate effect of calling createJob() is to push a job onto the queue, not to start its execution. A single client can't dictate the start time of a remote job because of the overall Server workload. But the client can expect createJob() to push a new job request on the queue in real time.
4. public void restartJobPhase2(JobID aJobID) throws RemoteException, Exception.
5. public Vector getJobs(String anRMIregistryName) throws RemoteException, Exception;
6. public void cancelJob(JobID aJobID) throws RemoteException, Exception;

If the job is queued, but not yet run, it is removed from the queue. If the job can't be found, an exception is thrown. If the server is quiet, (no job is running) then the variable currentJob will equal null. If it's non-null, then it could reference the job to cancel. If so, take care not to end the job when it is doing something important, such as updating the database or log file. To avoid such dirty writes, the cancellation technique works by setting a Boolean in the object currentJob. A job must, therefore, check this Boolean at appropriate times and stop itself gracefully if it has been cancelled by another thread. In this example, it throws a JobCancelled exception if the isCancelled Boolean is true.

Design of the Class ClientApplication

In this example, the class ClientApplication is just a tester for class ServerApplication. It is not complete or suggestive of how such a Client should be designed. However, it is important to note the RMI-related code.

The class is declared using two interfaces: Public class ClientApplication implements IR1clientApplication, java.io.Serializable.

The following are the important ClientApplication processing steps:

- Set the security manager.
- Construct the Server's URL.
- Construct an instance of class ClientApplication for export to the RMI registry.
- Export the client object reference to the RMI subsystem.
- Look up the Server's object reference in the RMI registry via its URL.
- Register the client with the server.
- Create jobs.

To create a Job object, all the necessary information is generated automatically by the clientApplication code. In this example, the user is prompted to enter a "1" to start a job on the server. If the user enters a "2" instead, the client then prompts the user to enter a "1" again to cancel the job. If the user enters a "2" instead, the code then prompts the user to enter another "1" to restart the job.

Design of Classes Job and Tjob

Job objects are constructed with three parameters. The first parameter – aClientList

– is a Vector of all ClientApplication objects the job should report to. For simplicity, the Server's jobs report to all connected clients.

The second parameter, JobRequest, comes unchanged from the ClientApplication object from the createJob() method. This object has all the information necessary to specify a correct job. The example includes things such as filenames to process, record time stamps, etc.

The TqManager parameter is a reference to the single TqManager object. A Job object needs this reference to unblock the thread when a job finishes.

```
void launchThread(){

  fTjob = new Tjob(this);
  fTjob.start();
} // end method
```

The launchThread method creates a new instance of the Tjob class and starts method Tjob.run(), shown below, which contains the driver code that calls methods in the Job object. Splitting a job's functionality between two classes – in this case Job and Tjob – was not necessary. It was done to aid human readability. For an excellent discussion of thread design issues, see *Java 1.1 Certification Study Guide*, S. Roberts and P. Heller, 1997, Chapter 7; and *Thinking in Java*, B. Eckel, Chapter 14, 1998)

Executing native code

Next is a discussion of several lines of the method JobrunPhase1(), because they show how to execute native code using the Runtime and Process classes. This method controls the execution of phase1.

The native process is executed using the Runtime class, which represents the JVM and Java interpreter. Besides executing native processes, Runtime objects also are used to get information about the host operating system.

```
  String command[] = null;
  int exitVal    = -1;
  command      = new String[2];
  try {
    if (fJobCancelled) {
      throw new JobCancelledException("Job was canceled
by client.");
    }// end if
    command[0] = ServerApplication.kBinPath;
    ServerApplication.log( "The native run command is["+
command[0] + "]" );

    // Execute the native module and wait

    processID = runtime.exec( command, fNativeEnvVars );
    this.setProcessID(processID);
    processID.waitFor();
```

The runtime.exec() method, shown above, executes the specified operating system command string. (For example, c:\\phase1.exe.) When you use Runtime to execute system commands, Process objects are created. The processID.waitFor() method, also shown above, causes the Tjob thread to block until phase1 finishes.

```
void setNativeEnvironmentVariables(){

  fNativeEnvVars[0] = "LIB=d:\\IBMPLIW\\LIB";
  fNativeEnvVars[1] = "LOCPATH=D:\\IBMPLIW\\LOCALE" ;
} // end method
```

The variable fNativeEnvironmentVariables, the setter of which is shown above, contains the native code's environment variables, which must be in an array of Strings. Note that in this example the two environment variables mentioned, LIB and LOCPATH are just dummies.

```
    BufferedReader inBufRdr;
    inBufRdr = new BufferedReader( new
InputStreamReader(processID.getInputStream()));
    String line=null;
    int err=0;
    while ((line = inBufRdr.readLine()) !=null) {
      ServerApplication.log( line );
      err = 1;
    }// end while
    if (err != 0) {
      String message = "On job " + this.identify() +", phase
2 "  + " failed, due to incorrect....???";
      ServerApplication.displayOnClients(message);
      throw new Exception("Native program failure.");
    }// end if
```

The processID.getInputStream method, shown above, gets error messages from the native process. The method's name is something of a misnomer because the input stream is connected to the output stream of the native process. Note that this stream is usually buffered.

```
    try {
      exitVal = processID.exitValue();
     } catch (Exception exptn) {
      String message = "Native program not found: " +
command[0];
      ServerApplication.abnormalEnd(new Exception(),mes-
sage);
    }// end catch
```

The method processID.exitValue, shown above, returns the exit value of the process. So if your native process is well-behaved, you can use its value to detect error information. However, this method is used only to catch the Exception, thrown when the JVM can't find the .exe file.

CONTINUED ON PAGE 12

CONTINUED FROM PAGE 11

```
protected void runPhase2()throws Exception{

  TableJobDetl tableJobDetl  = new TableJobDetl( ) ;
  tableJobDetl.insert(this);

} // end method()
```

The method runPhase2, shown above, simply creates a new instance of a database table object, then calls a method to insert some dummy job statistics into this table.

The design of Class TableJobMstr: JDBC techniques

Full coverage of JDBC techniques is beyond the scope of this article, but for an excellent discussion on the subject refer to *JDBC Database Access with Java*, Edition Number 01, Addison Wesley Longman, Incorporated, January 1997, ISBN: 0201309955, Authors: Hamilton, Graham / Cattell, Rick / Fisher, Maydene. Also IBM VisualAge for Java provides some excellent JDBC convenience classes.

```
public class TableJobDetl{
 private String fTableName;
 private Statement fStatement;
```

Class TableJobDetl illustrates the most important JDBC techniques. As you can see from its constructor below, when the Server needs to access a table, at least two references are needed: the table name, and the JDBC statement.

```
 TableJobDetl()throws Exception {

  fTableName = ServerApplication.fDBQualifier + "." +
"PROVDR_VALU_DETL";
  fStatement =
ServerApplication.fDBConnection.createStatement();
 } // end constructor
```

Statement objects, which are used to execute static SQL commands, are returned by the method java.sql.Connection.createStatement.

The method insert illustrates several useful JDBC techniques. This method reads records from a file and writes rows to a database table. The most important technique here is the use of java.sql.PreparedStatement. Instances of this class contain pre-compiled SQL statements. Use the PreparedStatement class, rather than the Statement class for better performance.

A second useful technique is the use of Java's StringTokenizer class, which contains methods to parse strings. In the example, StringTokenizer is used to help read the information in a flat file.

A third useful technique is the use of the java.sql.SQLException class to determine the exact type of SQL exceptions thrown. Note that the SQLState value is platform dependent.

A final useful technique shown in the insert method (***Listing 1***) is the use of the finally block to make sure prepared statements are always closed. Failure to explicitly close SQL statements can cause memory leaks because of Java's unpredictable garbage collection. Fixing these leaks on a big server can be difficult.

Conclusion

This article has presented the highlights of a proven, object-design for a medium-sized Java RMI native code server. The entire article, sample code and a list of the samples classes are available on the Developer Connection Web site at www.developer.ibm.com/devcon/.

Keith Wright's *career with IBM started at Tivoli as a course developer in May 1997. Currently, he is an architect in the Global Business Intelligence Solutions (GBIS) unit, building Java software solutions in data warehousing and data mining. He has been a developer for more than 15 years. He earned his Ph.D. in Business Information Systems at UT Austin. You can contact him at mkwright@us.ibm.com.*

```
 void insert(Job aJob)  throws Exception{

  JobID jobID  = aJob.getJobID();
   String delimiter = ",";
  String idEntity;
  String idFeat;

  String sqlStatement = "INSERT INTO " + fTableName  + " VAL-
UES ( ?, ?, ?, ?, ?)";
  String line=null;
  String token;
  PreparedStatement prepdInsertStmt=null;
  FileInputStream fileInputStream;
  InputStreamReader inputStreamReader;
  BufferedReader bufferedReader = null;
  try {
    prepdInsertStmt =
ServerApplication.fDBConnection.prepareStatement(
sqlStatement);
    fileInputStream  = new FileInputStream("e:\\sandbox\\arti-
cle\\serverInputData.txt");
    inputStreamReader  = new
InputStreamReader(fileInputStream);
    bufferedReader  = new BufferedReader(inputStreamReader);


    // Read and write loop
    while ((line = bufferedReader.readLine()) !=null) {
     try{
      StringTokenizer stringTokens =
        new StringTokenizer(line, delimiter);
        // Check to see if Phase was canceled
        if (aJob.isCancelled() ){
          throw new JobCancelledException("");
        }// end if
        idEntity = stringTokens.nextToken();
         prepdInsertStmt.setString(1,  "LACHIR" );
         prepdInsertStmt.setString(2,  "AA" );
         prepdInsertStmt.setString(3, "smith01" );
         prepdInsertStmt.setString(4, "bp0001" );
         prepdInsertStmt.setDouble(5,  1.111);
        prepdInsertStmt.executeUpdate();

      }// end inner try
      catch (SQLException sqlExptn ) {
        String sqlState = sqlExptn.getSQLState();
        if((sqlState.equals("23505")
          && (aJob instanceof JobRestartPh2))){
          // this state means duplicate primary key
          //System.out.print("d");
        }// end if
        else{
          throw sqlExptn;// re-throws to fatal catch below
        }// end else
      }// end catch
    }// end while
  }// end outer try
  catch (JobCancelledException exptn2) {
    throw exptn2;
  }// end catch
  catch (FileNotFoundException exptn3) {

    String message = "Phase 2 input file not found.";
    throw new JobPhase3Exception(message);
  }// end catch
  catch (Exception exptn4 ) {

   String message = "Problem inserting records in database";

  }// end catch
  finally{
    if(prepdInsertStmt!= null){
      prepdInsertStmt.close();
    }// end if
    bufferedReader.close();
  }// end finally
 }// end method
```

Listing 1.

Signing and verifying with certificates in Java

by Theodore Shrader, Bruce Rich, and Anthony Nadalin

In our last article, "On the Trail of Certificates in Java 2," we discussed certificates, their representation in Java, and their importance in e-commerce. This article delves deeper into the use of certificates in Java, particularly their use in signing data and verifying signatures. A later example demonstrates how to create a signature for a message and verify the authenticity of the signature using classes that are part of the Java Runtime Environment (JRE).

Overview of the signing process

As noted in the previous article, certificates represent the public identity of a user or entity as granted by a certificate authority. Certificates also exploit the public key architecture. In this model, a selected algorithm will generate a public and private key pair. The key pair holds a special relationship. Data that has been encrypted by a private key can be decrypted only by the accompanying public key. The reverse relationship also is true. Data encrypted by a public key can be decrypted only by the associated private key. After a key pair is generated for a user, the public key is stored within the user's public certificate and the private key is squirreled away and kept hidden by the user. These public and private key pairs play a pivotal role in signing data and verifying signatures.

Signing raw data or data representations of objects involves two steps. First, the signing process produces a message digest of the data. Next, the signing process encrypts the digest with the private key to create a signature of the data. When the sender transmits data with its generated signature to another user, the recipient can verify the signature with the public key that is associated with the private key used in signing. Commonly signed data objects include applet JAR files and e-mail messages.

Figure 1 shows the steps of the signing process in more detail.

Message digest

A generated message digest represents a unique hash code for the digested data. Algorithms that create message digests endeavor to ensure that the generated digest is unique, so that even the slightest change to the original data causes the new digest to be different from the old. The algorithms also ensure the digest itself cannot be used as part of a reverse engineering process to discover the original data.

Popular message digest algorithms include SHA-1 and MD5, although the use of MD5 has waned in favor of the stronger SHA-1 algorithm. The MD2 and MD4 algorithms are outdated and should not be used, due to their short key length or vulnerability.

Message digests help recipients verify the integrity of the digested data. As useful as message digests are, they cannot be used for authentication. If a user receives data and its accompanying digest, the user can use the same message digest algorithm on the data to compute the digest and compare the computed digest against the received digest. If they match, users can be sure that the data has not changed. However, users cannot be certain that the sender of the data and digest is bona fide. The transformation of the digest to a signature value allows the recipient to authenticate the source of the data.

To make e-commerce successful, businesses and developers must be able to sign objects to state that these objects originated from them.

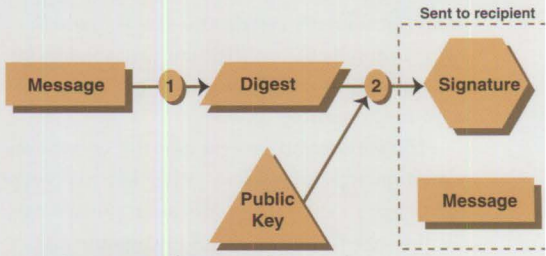


Figure 1. Signing process

Generating signatures

To add authentication to the signature process, the sender must put the digest through another step where the digest is encrypted with the sender's private key. The result of the encryption is a signature, an encrypted digest. For the sender to allow the recipient to authenticate the received data, the sender would need to send the data to the recipient with the data's signature. Note that the digest was encrypted and not the message itself.

Popular signature algorithms include DSA, which is the default supplied with Java, and RSA, which is supported by various entities, such as Netscape Web browsers. The signature algorithm attribute on a certificate typically is identified with the combination of the message digest and encryption algorithms. For example, "SHA1withRSA" indicates that the signature for the accompanying data was generated by creating a message digest of the data with the SHA1 algorithm and encrypting the digest with the RSA algorithm.

Verifying signatures

To verify a signature, the recipient needs the data, its signature and the certificate of the sender. Since the sender transmitted it, the recipient already has the data and its signature. The recipient can retrieve the sender's certificate as part of an accessible certificate database or LDAP directory, for example. With only

CONTINUED ON PAGE 14

CONTINUED FROM PAGE 13

these three objects, the recipient can verify the integrity of the data as well as the authenticity of the sender.

Figure 2 shows the steps of the verification process in more detail.

The verification process begins by extracting the signature algorithm and public key from the sender's certificate. The signature algorithm tells the verification process which message digest and encryption algorithm was used in the signing process.

The verification process uses the encryption algorithm and the sender's public key to decrypt the signature. Next, the verification process uses the message digest algorithm to compute an independent message digest of the data. Lastly, the process compares the independently computed digest with the decrypted signature. If the two are the same, verification succeeds and the recipient can sleep well knowing the data was authenticated and its integrity validated. If the two values differ, the recipient can be assured that something is wrong.

Java classes

The Java 2 JRE contains a framework for signing and verification and includes some implementations of public domain cryptography algorithms sufficient to allow developers to sign and verify data. The basic security classes that include support for message digests and signatures are part of the Java Cryptographic Architecture (JCA). However, the JRE does not contain classes to encrypt or decrypt data. Ciphers, key exchange algorithms and their accompanying classes are part of the Java Cryptographic Environment (JCE) package. The JCE is restricted for export and thus is packaged separately. The JCE also plugs into the JCA, and the JCA is structured so that other companies can provide additional packages that support other security classes, such as RSA signatures.

Generating a signature includes encryption, but since the encryption is restricted to the message digest of an object or data, the encryption can be part of the signature process that is included with the JRE. With the JRE classes, developers cannot encrypt or decrypt arbitrary data.

An example

The following example illustrates how developers can sign data and verify signatures within Java. For conciseness, both the signing and verification steps are performed in the same example program and not all the error checking has been included.

The signing and verification processes are typically broken into different modules or applications. The Public-Key Cryptography Standards (PKCS) #7 provides a standard in which data and its signature can be bundled into a SignedData object and transmitted to other heterogeneous applications. The Distinguished Encoding Rules (DER) encoded definition for SignedData includes the data, signature and identity of the sender. The recipient would need to look up the identity in a certificate database to retrieve the certificate that contains the sender's private key. The DER encoding of the SignedData object could be delivered to the recipient application via a socket or file, for example.

In the following example, Benjamin wants to send the following important message to Patrick: "Lee and McClellan are at Antietam." Ben is not concerned about encrypting the message but wants Patrick to know that the message originated from Ben and that no one modified the message in transit.

This example uses the default Java keystore that handles the DSA signature format for its certificates. The program loads the keystore and obtains Ben's certificate before extracting the public key, which is used in the verification process. In a separate step, the program also retrieves the private key, which would only be used in the signature process and only be known to Ben.

To generate a signature, the program creates a message digest of the data. The JCA classes work similarly in that the calling program first must get an instance of the class by specifying the desired algorithm with the getInstance static method. In this program, after obtaining an instance of the MessageDigest class, the program feeds the message into the MessageDigest instance and generates the digest using the digest method.

Next, the program uses the message digest and Ben's private key to generate a signature. As before, the program gets an instance of the Signature class using the getInstance static method. Before the Signature instance can be used for signing, it must be initialized with the initSign method that takes Ben's private key as an argument. The program feeds the digest into the Signature instance with the update method and retrieves the signature with the sign method.

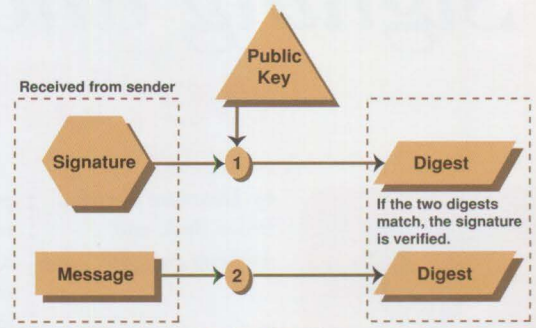


Figure 2. Verification process

The verification process is similar, in that the program first must get an instance of the Signature class, but to initialize it, the program instead calls the initVerify method with Ben's public key. The digest is fed into the Signature instance as before, but now the program calls the verify method with the signature that accompanied Ben's message as input. The verify method returns true if the decrypted signature matches the computed digest.

Thus, if Patrick wanted to verify a message and signature that Ben sent, Patrick could call the verifySignature method in this example from his own module. If Patrick received a return of true from the method with the first digest, Patrick would know that it was his friend Ben who sent the message. If Patrick received false from the method, such as with the fake message, Patrick would know that the message did not come from Ben or that the message was changed in transit.

Note that the JCA will map the DSA signature algorithm to the SHA1withDSA algorithm. The example in this article illustrates the steps separately. You could send a byte array of the message directly to the Signature instance under the aegis of the SHA1withDSA algorithm as part of the signature or verification process.

Listing 1 shows the example program and *Figure 3* displays the output.

Conclusion

To make e-commerce successful, businesses and developers must be able to sign objects to state that these objects originated from them. In turn, consumers and users must be able to verify that objects they receive came from their professed owners and arrived without modification. By following the role of public and private keys and specified algorithms, we have demystified the signing and verification process.

For more information

- For an overview of encryption in Java, consult the companion article, "Java Cryptography (Part 1: Encryption and Decryption)," elsewhere in this magazine.
- For more information on Java Security, consult the JavaSoft Web site at java.sun.com/products/jdk/1.2/docs/guide/security/index.html.
- For more information on the PKCS standards, consult the RSA Web site at www.rsa.com/rsalabs/pubs/PKCS/.

Theodore Shrader *is a feature lead in the IBM Java Security project. He has written numerous patents and articles dealing with Internet and Java development, distributed computing, object-oriented design and database architecture and programming. He also is a co-author of an operating systems programming guide published by John Wiley and Sons. You can contact him at tshrader@us.ibm.com.*

Bruce Rich *is the team lead of the IBM Java Security project. He has been involved in software for 19 years, first in operating systems development, then in secure distributed file systems, more recently in secure Web server applications and Java. He has filed a number of patents and contributed to a book on distributed computing. You can contact him at rbruce@us.ibm.com.*

Anthony Nadalin*, currently, is the Lead Architect for Java Security. As senior architect for Java Security, he has responsibility for infrastructure design and development across IBM. He serves as the primary security liaison to JavaSoft for security design and development collaboration. You can contact him at drsecure@us.ibm.com.*

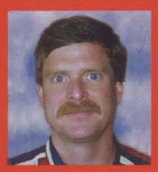

```
package signtest;

import java.io.*;
import java.security.*;
import java.security.cert.*;

public class SignAndVerify {

   String message = "Lee and McClellan are at Antietam.";
   String fakemessage =
     "Lee is at Richmond and McClellan is at Washington.";

   String keystoreFilename =
     "C:\\WINNT40\\Profiles\\Administrator\\.keystore";
   String certificateAlias = "benalias";

   String privateKeyName = "benalias";
   String privateKeyPassword = "BenPwd";

   String signatureAlgorithm = "DSA";
   String digestAlgorithm = "SHA1";

   X509Certificate x509cert;
   PublicKey publicKey;
   PrivateKey privateKey;

   public static void main(String args[]) {
     SignAndVerify prog = new SignAndVerify();
     prog.run();
   }

   public void run() {
     try {

     System.out.println("Loading the keystore and retrieving keys.");

     KeyStore ks = KeyStore.getInstance("JKS"); // using the JavaSoft
         keystore

     FileInputStream ksinputstream = new FileInputStream(keystoreFilename);
         ks.load(ksinputstream, null);

     x509cert = (X509Certificate) ks.getCertificate(certificateAlias);

     publicKey = x509cert.getPublicKey();

     privateKey = (PrivateKey) ks.getKey(privateKeyName,
       privateKeyPassword.toCharArray());

     System.out.println("Signing message:\r\n\t" + message);

     byte[] digest = this.getMessageDigest(message);
     byte[] signature = this.getSignature(digest, privateKey);

     System.out.println("Verifying signature with the real message.");

     boolean bverify = this.verifySignature(digest, signature, publicKey);
     if (bverify) {
       System.out.println("SUCCESS: Signature verified.");
     } else {
       System.out.println("ERROR: Signature did not verify.");
     }

     System.out.println("Verifying signature with the fake message:"
       + "\r\n\t" + fakemessage);

     byte[] fakedigest = this.getMessageDigest(fakemessage);

     bverify = this.verifySignature(fakedigest, signature, publicKey);
       if (bverify) {
         System.out.println(
           "ERROR: Signature verified with the fake message.");
       } else {
         System.out.println(
           "SUCCESS: Signature did not verify with the fake message.");
       }

       } catch (Exception e) {
         System.out.println("*** Error: " + e);
         System.exit(0);
       }

     }

     public byte[] getMessageDigest(String message) {
       try {
         MessageDigest md =MessageDigest.getInstance(digestAlgorithm
         return md.digest(message.getBytes()); // generate message
             digest
       } catch (Exception e) {
         System.out.println("*** Error: " + e);
         System.exit(0);
       }
       return null;
     }

     public byte[] getSignature(byte[] digest, PrivateKey privateKey) {
       try {
         Signature sign = Signature.getInstance(signatureAlgorithm);

         // Initialize the signature object with the private key for
             // signing.

         sign.initSign(privateKey);
         sign.update(digest); // feed in the digest
         return sign.sign(); // generate the signature
       } catch (Exception e) {
         System.out.println("*** Error: " + e);
         System.exit(0);
       }

       return null;
     }

     public boolean verifySignature(byte[] digest, byte[] signature,
       PublicKey publicKey) {

       try {
         Signature sign = Signature.getInstance(signatureAlgorithm);

         // Initialize the signature object with the public key for
             // verifying.

         sign.initVerify(publicKey); // feed in the digest
         sign.update(digest); // test with the signature
         return sign.verify(signature);
       } catch (Exception e) {
         System.out.println("*** Error: " + e);
         System.exit(0);
       }

       return false;
     }
}
```

Listing 1. Example program

```
c:\>java signtest.SignAndVerify
Loading the keystore and retrieving keys.
Signing message:
        Lee and McClellan are at Antietam.
Verifying signature with the real message.
SUCCESS:  Signature verified.
Verifying signature with the fake message:
        Lee is at Richmond and McClellan is at Washington.
SUCCESS:  Signature did not verify with the fake
message.
```

Figure 3. Example output program

The Security Migration Aid: *Bringing Java 2 security* to Java 1.1.8


by Bruce Rich, Anthony Nadalin and Theodore Shrader

In July of 1999, IBM made the following available:

- IBM Runtime Environment for Windows,▲ Java Technology Edition, Version 1.1.8,
- IBM OS/2* Warp Runtime Environment, Java Technology Edition, Version 1.1.8, and
- IBM OS/390 Runtime Environment, Java Technology Edition, Version 1.1.8.

For developers, IBM also released the following:

- IBM Developer Kit for Windows, Java Technology Edition, Version 1.1.8,
- IBM OS/2 Warp Developer Kit, Java Technology Edition, Version 1.1.8, and
- IBM OS/390 Developer Kit, Java Technology Edition, Version 1.1.8.

Note: The Java Development Kits (JDKs) are referred to as Standard Development Kits (SDKs) in Java 2.

For the first time, the IBM products consist of more than just the base runtime environment or developer kit content. Available with each offering are Swing 1.1, Remote Method Invocation over Internet Inter-ORB (Object Request Broker) Protocol (RMI-IIOP) and a security migration aid. These have been integrated into the IBM test regime so developers can incorporate them into their products with the knowledge that they have been tested by and are supported by IBM. These packages vary slightly by platform.

This article presents an overview of the Security Migration Aid: what it is; why it was created; and how you can exploit it.

What is it?

The combination of the core Java classes plus a Java Virtual Machine (JVM) specific to each OS platform is called a Java Runtime Environment (JRE). The core classes act as a sometimes thick, sometimes thin Java skin over the top of the JVM, largely written in C. An important part of the Java 2 content was the new policy-based security model. The Security Migration Aid is a port of the Java 2 security classes back onto the 1.1.8 version of the Java Virtual Machine. IBM has provided this capability as a separate package, so that customers may choose to run their applications on the unmodified 1.1.8 runtime environment or to augment their application's behavior with the Security Migration Aid capabilities.

Why provide a Security Migration Aid?

One of Java's salient features has been its contribution to secure computing. Its ability to run applets in a sandbox that kept Internet-based attacks at arm's length contributed to its wide acceptance in the beginning. Now that Java is being used on back-end servers inside many enterprises, it is just as important to limit the damage that an errant Java application or servlet might do. The new Java 2 security code allows that capability and does it through a fine-grained policy-based implementation that is both useful and extensible.

IBM is offering the new security model over the top of the Java Virtual Machine (JVM) in the JDK Version 1.1.8, so that customers can exploit a tried and true, high-performance JVM, and, at the same time, start to move Java applications into the up and coming security framework for Java.

The Java 2 platform security framework is an impressive step forward for secure computing in Java.

What's the Java 2 security model?

So, what is implied by the assertion that the IBM 1.1.8 Java products have a separate package that contains the Java 2 security classes? For a discussion of the security model and classes, consult http://java.sun.com/products/jdk/1.2/docs/index.html for the official Java 2 SDK Standard Edition V1.2 online application programming interface documentation, or http://java.sun.com/products/jdk/1.2/docs/guide/security/index.html for extensive information on just the security framework.

As a summary of the implementation and its implications, Java 2 has a concrete implementation of a SecurityManager. This new SecurityManager consults an external security policy in making its access decisions, and allows application- or user-defined Permissions, so it is flexible and extensible. Setting the system's security manager to the default SecurityManager by this code sequence

```
System.setSecurityManager(new java.lang.SecurityManager());
```

sets in motion a policy-based, more tightly controlled environment in which to run Java programs.

Java 1.x has always had a SecurityManager class, contained in java.lang.SecurityManager. This SecurityManager was abstract, so usage would have required subclassing. Even though the SecurityManager was abstract, it had implementations at every check method that would throw a SecurityException, so subclassing would require overriding every check method that did not always throw an exception. Furthermore, there was no external formalized policy that one could use to control the JVM's actions, so each application that wished to enforce security had to implement much of its own framework for security permission specification.

How is the Security Migration Aid different from the real thing?

Since there are no changes in the native code for the JVM in version 1.1.8 to support the Security Migration Aid, there are some limitations in the support that can be provided. For example, there are some command line options that one can specify when launching applications in Java 2 that the Java 1.1.8 commands ignore. Also, there are some new capabilities that Java 2 would be able to control (for example, the methods that the new java.lang.reflect.ReflectPermission classes control) that the JDK 1.1.8 is unaware of, so it cannot enforce. These differences are minor.

There is a *Security Migration Aid User's Guide* that is available with the IBM 1.1.8 JVM platform distribution that goes into more detail. The URLs to the appropriate OS platform JDKs are given at the end of this article. The following illustrate some key points derived from our Windows implementation:

- Launching an application under the Java 2 default SecurityManager

  In Java 2:

```
java -Djava.security.manager someApplication
```

  Under the Security Migration Aid:

```
java com.ibm.security12.sun.misc.Main someApplication
```

- Launching an applet under the Java 2 concrete SecurityManager

  In Java 2:

```
appletviewer xxx.html
```

  Under the Security Migration Aid:

```
java com.ibm.security12.sun.applet.AppletViewer xxx.html
```

Notice that the invocation in the second illustration did not just insert a layer of additional Java between the command and the applet. In this case, the actual command had to change, as the appletviewer command is hardwired to a particular package, sun.applet, rather than the com.ibm.security12.sun.applet package that we wish it to use.

In both examples, we inserted some additional Java code between the native command ("java") and the Java program we wish to actually run (either "someApplication" or whatever is specified in xxx.html). This additional layer of Java provides the functionality that is missing from the 1.1.8 JVM, among other things. The second illustration shows that the fine-grained, policy-based Java 2 security framework applies to applets as well. Thus the Security Migration Aid also can assist in getting applets ready for the Java 2 environment, as well as applications.

What else do I need to know?

There are a few other points that need to be made regarding the Security Migration Aid. First, if developers begin looking inside the Security Migration Aid packages, they will find most of the Java 2 Collection classes in the com.ibm.security12.java.util package. This was done primarily because the security classes, themselves, were using the new Collection classes, so it was easier to include the additional classes than re-work the security code. Programmers may be tempted to avail themselves of the Collection classes inside the Security Migration Aide packages. However, if programmers make use of these Collection classes in their code, they could have a migration issue when going to a real Java 2 platform, as their code is looking in the Security Migration Aid packages rather than in the normal java.util package.

Second, there is some additional information that the Security Migration Aid needs to know to operate correctly. To minimize migration issues, both from Java 1.1.x and to Java 2, the Security Migration Aid supports the use of companion files for storing this additional information. This information could be put into configuration files that the 1.1.8 runtime already looks at, e.g., java.security, but doing so would complicate migration to Java 2, so use of the companion files is encouraged wherever possible. The names of these companion files will differ on different platforms, so please consult the relevant platform documentation for more information.

Lastly, this article is simply intended as an overview. For more information, consult the documentation included with the IBM Java 1.1.8 distributions.

Conclusion

The Java 2 platform security framework is an impressive step forward for secure computing in Java. The Security Migration Aid allows developers to begin migrating their applications to the new security framework without prematurely involving all the other new features of Java 2.

Additional reading

Gong, Li. *Inside Java 2 Platform Security.* Reading, MA: Addison Wesley, 1999.

Pistoia, Marco, et al. *Java 2 Network Security.* Research Triangle Park, NC: International Business Machines Corporation, 1999.

For information on the IBM OS/390 Developer Kit, Java Technology Editions, Version 1.1.8 or the IBM OS/390 Runtime Environment, Java Technology Editions, Version 1.1.8, see http://www.s390.ibm.com/java/about118.html.

For information on the IBM OS/2 Warp Developer Kit, Java Technology Edition, Version 1.1.8 or the IBM OS/2 Warp Runtime Environment, Java Technology Editions, Version 1.1.8, see http://techsupport.services.ibm.com/asdbin/doc/en_us/java/f-feat.htm.

For information on the IBM Developer Kit for Windows, Java Technolgoy Edition, Version 1.1.8 or the IBM Warp Runtime Environment for Windows, Java Technology Editions, Version 1.1.8, see http://www.ibm.com/java/jdk/118.

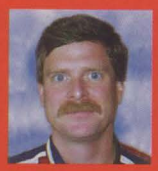
Bruce Rich *is the team lead of the IBM Java Security project. He has been involved in software for 19 years, first in operating systems development, then in secure distributed file systems, more recently in secure Web server applications and Java. He has filed a number of patents and contributed to a book on distributed computing. You can contact him at rbruce@us.ibm.com.*

Anthony Nadalin *currently is the Lead Architect for Java Security. As senior architect for Java Security, he has responsibility for infrastructure design and development across IBM. He serves as the primary security liaison to JavaSoft for security design and development collaboration. You can contact him at drsecure@us.ibm.com.*

Theodore Shrader *is a feature lead in the IBM Java Security project. He has written numerous patents and articles dealing with Internet and Java development, distributed computing, object-oriented design and database architecture and programming. He also is a co-author of an operating systems programming guide published by John Wiley and Sons. You can contact him at tshrader@us.ibm.com.*

Java cryptography

Part 1: Encryption and decryption



by Anthony Nadalin, Theodore Shrader and Bruce Rich

Beginning with Version 1.1 of Sun's® Java Development Kit (JDK), general purpose APIs for cryptographic functions, collectively known as the Java Cryptography Architecture (JCA), are provided along with its extension, the Java Cryptography Extension (JCE). The Java 2 Standard Development Kit (SDK) significantly enhances the Java Cryptography Architecture. Specifically, the Java 2 SDK augments the certificate management infrastructure to support X.509 V3 certificates.

Note: The Java Development Kits (JDKs) are referred to as Standard Development Kits (SDKs) in Java 2.

This article describes the basic classes used in encryption and decryption and how to create a simple program exploiting these features. Additional articles are planned that delve deeper into advanced topics about the Java Cryptography Extension (JCE).

United States export considerations

The security classes shipped with the Java 2 Standard Development Kit provide for only the message digest and digital signature part of the cryptographic spectrum. This allows you to perform reliable authentication that, in turn, can be used as a basis for implementing access controls that relax the sandbox restrictions. However, the Java Cryptography Architecture alone does not provide the general purpose encryption functionality needed to send confidential data.

This function is provided by the JCE, which is an extension to the cryptography-related classes shipped with the Java 2 SDK. JCE uses the same structure as the JCA, being composed of engine classes that expose the algorithms in a generic way. The JCE provides engine classes for (a)symmetric-key encryption and for generating and manipulating the secret keys that such algorithms require.

The primary principle in the design of the JCA has been to separate the cryptographic concepts from their algorithmic implementations. The separation of the encryption classes satisfies the current United States Export requirements and allows the SDK with the JCA to be freely distributed, while restricting the distribution of the JCE. Before we explain how JCA achieves this separation, it is worthwhile to review the types of classes supplied by the Java 2 SDK, the APIs that are part of the JCA and the API extensions supplied by the JCE.

Java 2 SDK, JCA and JCE APIs relationship

The Java 2 SDK APIs consist of core classes that are shipped with the Java Virtual Machine (JVM). The set of core classes in the Java 2 platform can be divided into two subsets:

- Security-related core classes
- Other core classes

The security-related core classes can be further subdivided as:

- Access control and permission related core classes
- Cryptography-related core classes

Of these, only the cryptography-related core classes are part of the JCA APIs. The JCE extends the JCA API to include APIs for encryption, key exchange, and Message Authentication Code (MAC). Together, the JCE and the cryptography aspects of the Java 2 SDK provide a platform-independent cryptography API. The JCE is released separately as an extension to the Java 2 SDK, in accordance with current United States export control regulations.

The primary principle in the design of the JCA has been to separate the cryptographic concepts from their algorithmic implementations.

Java Cryptography Extension 1.2

JCE has been provided as an extension to the Java platform. JCE 1.2 provides a framework and implementation for encryption, key generation, key agreement, and Message Authentication Code (MAC) to supplement the interfaces and implementations of message digests and digital signatures provided by Java 2 SDK, Standard Edition, V1.2.

The provider architecture of the JCA aims to allow algorithm independence. The design principles behind JCE also share the same philosophy of implementation and algorithm independence by the use of the provider architecture. In addition to making it possible to use newer algorithms for generating keys, JCE also introduces some very new interfaces and classes that facilitate the implementation of these concepts.

JCE provides a framework for encryption, (session) key generation and key agreement, and Message Authentication Code (MAC) algorithms. Support for encryption includes symmetric, asymmetric, block, and stream ciphers.

Provider and the packages

The SunJCE provider consists of the main package javax.crypto and its two subpackages javax.crypto.spec and javax.crypto.interfaces.

The javax.crypto package forms the main body of the JCE 1.2 class structure. The package primarily consists of classes that represent the new concepts of ciphers, key agreements, and message authentication codes and their corresponding SPI classes.

The javax.crypto.spec package consists of various key specification and algorithm para-

meter specification classes.

The javax.crypto.interfaces package consists of the DHKey interface and a couple of its subinterfaces – DHPrivateKey and DHPublicKey. These are the interfaces for the keys based on the Diffie-Hellman algorithms.

The Cipher Class

The javax.crypto.Cipher engine class forms the core of the JCE 1.2 framework. This class provides the functionality of a cryptographic cipher used for encryption and decryption.

Like other engine classes, the Cipher class is instantiated using its getInstance() factory method. This method takes as argument a String object that represents a transformation. A transformation is a string that describes the operation (or set of operations) to be performed on the given input, to produce some output. A transformation always includes the name of a cryptographic algorithm (for example, Data Encryption Standard or DES), which may be followed by a feedback mode and padding scheme:

- A Cipher object obtained from getInstance() must be initialized for either encryption or decryption mode. These modes are defined as final integer constants in the Cipher class. The two modes can be referenced by their symbolic names ENCRYPT_MODE and DECRYPT_MODE.
- Algorithms usually operate on blocks having a predefined size. Plain text packets, which are of a length not a multiple of that size, must be padded prior to encrypting according to a specified padding scheme.

From what we said, a transformation is of the form algorithm/mode/padding or algorithm. If mode and padding are not specified, provider-specific default values are used.

For example, the following is a valid way to create a Cipher object:

```
Cipher c = Cipher.getInstance("DES/CBC/PKCS5Padding");
```

Optionally, getInstance() can accept, as a second argument, the name of the provider after the transformation parameter.

A Cipher object is initialized by calling the init() method. When this happens, it loses all previously acquired states. In other words, initializing a Cipher is equivalent to creating a new instance of that Cipher and initializing it.

Data can be encrypted or decrypted in one step (single-part operation) or in multiple steps (multiple-part operation). You will encrypt or decrypt data in a single step or in multiple steps depending on whether you call the doFinal() or update() method, respectively. A multiple-part operation is useful if the exact length of the data is not known in advance, or if the data is too long to be stored in memory all at once.

The Cipher stream classes

JCE 1.2 introduces the concept of secure streams, which combine an InputStream or OutputStream with a Cipher object. Secure streams are provided by the CipherInputStream and CipherOutputStream classes:

- The javax.crypto.CipherInputStream class is a FilterInputStream that encrypts or decrypts the data passing through it. It is composed of an InputStream, or one of its subclasses, and a Cipher.

  CipherInputStream represents a secure input stream into which a Cipher object has been interposed. The read methods of CipherInputStream return data that are read from the underlying InputStream but have additionally been processed by the embedded Cipher object.

  Notice that the Cipher object must be fully initialized before being used by a CipherInputStream. For example, if the embedded Cipher has been initialized for decryption, the CipherInputStream will attempt to decrypt the data it reads from the underlying InputStream before returning it to the application.
- The javax.crypto.CipherOutputStream class is a FilterOutputStream that encrypts or decrypts the data passing through it. It is composed of an OutputStream, or one of its subclasses, and a Cipher.

  CipherOutputStream represents a secure output stream into which a Cipher object has been interposed. The write methods of CipherOutputStream first process the data with the embedded Cipher object before writing it out to the underlying OutputStream.

The Cipher object must be fully initialized before being used by a CipherOutputStream. For example, if the embedded Cipher has been initialized for encryption, the CipherOutputStream will encrypt its data, before writing it out to the underlying output stream.

Sample program using the JCE 1.2 APIs to encrypt and decrypt

Listing 1 shows a simple program, EncryptDecrypt.java, that reads data from an

CONTINUED ON PAGE 20

CONTINUED FROM PAGE 19

input file; encrypts it using a Cipher object, jcecipher1, initialized for the DES algorithm in the ENCRYPT_MODE; decrypts it using another Cipher object, jcecipher2, initialized for the DES algorithm in the DECRYPT_MODE; and then prints the output to a file.

The output file name also is specified on the command line.

Use the following comand to compile:

```
javac EncryptDecrypt.java
```

For the above command to work, the SunJCE provider must be installed on your system. Make sure your CLASSPATH environment includes the JCE classes.

Then launch the program, passing on the command line the name of an existing input text file, jce.txt, and the name of an output file where the EncryptDecrypt program writes the data that it has encrypted and decrypted:

```
java EncryptDecrypt jce.txt jce_out.txt
```

To check the results, open the jce_out.txt file with a text editor, and verify that its contents are exactly the same as the contents of the original file jce.txt.

Conclusion

Java 2 security-related classes are divided between those available with the Standard Development Kits and the Java Cryptography Extension (JCE). The JCE includes classes that extend the Java Cryptography Architecture and allow pluggable cryptographic classes. With the JCE, developers can harness the power of encryption in their applications.

For more information on the signing and verification process, including the role of message digests, consult the companion article, "Signing and Verifying with Certificates in Java," elsewhere in this magazine.

For more information on Java Security, consult the JavaSoft Web site at java.sun.com/products/jdk/1.2/docs/guide/security/index.html.

Anthony Nadalin *is the lead architect for the IBM Java Security project. As the senior architect, he has responsibility for infrastructure design and development across IBM. He serves as the primary security liaison to JavaSoft for security design and development collaboration. You can contact him at drsecure@us.ibm.com.*

Theodore Shrader *is a feature lead in the IBM Java Security project. He has written numerous patents and articles dealing with Internet and Java development, distributed computing, object-oriented design, and database architecture and programming. He also is a co-author of an operating systems programming guide published by John Wiley and Sons. You can contact him at tshrader@us.ibm.com.*

Bruce Rich *is the team lead of the IBM Java Security project. He has been involved in software for 19 years, first in operating systems development, then in secure distributed file systems, more recently in secure Web server applications and Java. He has filed a number of patents and contributed to a book on distributed computing. You can contact him at rbruce@us.ibm.com.*

```
import java.io.*;
import java.security.*;
import javax.crypto.*;

class EncryptDecrypt {
  public static void main(String args[])
  {
   if (args.length != 2)
     System.out.println("Usage: java EncryptDecrypt inputFileName
outputFileName");
   else
   {
    try
    {
     // generate Cipher objects for encoding and decoding
     Cipher itsocipher1 = Cipher.getInstance("DES");
     Cipher itsocipher2 = Cipher.getInstance("DES");

     // generate a KeyGenerator object
     KeyGenerator KG = KeyGenerator.getInstance("DES");
     System.out.println("Using algorithm " + KG.getAlgorithm());

     // generate a DES key
     Key mykey = KG.generateKey();

     // initialize the Cipher objects
     System.out.println("Initializing ciphers...");
     itsocipher1.init(Cipher.ENCRYPT_MODE, mykey);
     itsocipher2.init(Cipher.DECRYPT_MODE, mykey);

     // creating the encrypting cipher stream
     System.out.println("Creating the encrypting cipher stream...");
     FileInputStream fis = new FileInputStream(args[0]);
     CipherInputStream cis1 = new CipherInputStream(fis, itsocipher1);

     // creating the decrypting cipher stream
     System.out.println("Creating the decrypting cipher stream...");
     CipherInputStream cis2 = new CipherInputStream(cis1, itsocipher2);

     // writing the decrypted data to output file
     System.out.println("Writing the decrypted data to output file " + args[1]);
     FileOutputStream fos = new FileOutputStream(args[1]);
     byte[] b2 = new byte[1024];
     int i2 = cis2.read(b2);
     while (i2 != -1)
     {
       fos.write(b2, 0, i2);
       i2 = cis2.read(b2);
     }

     fos.close();
     cis1.close();
     cis2.close();
    }
    catch (Exception e)
    {
     System.out.println("Caught exception: " + e);
    }
  }
}
```

Listing 1.

Enterprise JavaBean business components

by James Carey and Paul Monday

In order to dramatically increase the speed in which business applications are developed, domain knowledge (often called business content) must be captured and provided in a reusable component. The IBM Enterprise JavaBean Business Component initiative is building this type of business component. Building components which encapsulate domain knowledge using the Enterprise JavaBean specification enable components to be portable and to work together through common interfaces.

This article discusses the advantages of using business components built following the Enterprise JavaBean specification, the type of content involved in the IBM Enterprise JavaBean Business Component initiative, as well as how we arrive at that content. It also discusses why a person would choose to use one of these business components versus a similar component from another component model, such as a component framework.

Business components using Enterprise JavaBeans

The Enterprise JavaBean (EJB) component model brings about an important change in application development. In the past, developers were forced to consider what is best termed as infrastructure. Infrastructure consists of object distribution, transaction management, storage management, and more. Sun partitioned the infrastructure job from the component developer job in the Enterprise JavaBean specification through the definition of EJB Containers. Upon completion of an EJB business component, the developer deploys the component into a container. The container is then responsible for the infrastructure capabilities.[1] This split allows component developers to focus on business logic and lets container providers worry about the infrastructure.

In addition to partitioning the job of providing infrastructure capabilities from the job of assembling and building of business logic, the Enterprise JavaBean component model allows components from multiple vendors to work together seamlessly.

For example as shown in ***Figure 1***, take two traditional applications: one built to do inventory management for a parts supplier and another to do distribution management system for a parcel delivery company. Much of each application is tailored to infrastructure concerns that are usually tied to a particular operating system and storage capability (database). Suppose the parts supplier purchases the parcel delivery company for their distribution capabilities. Combining the original applications will be difficult to achieve. In fact, it may not even be possible if the applications started with different base assumptions about what type of transaction management, storage solution, and operating environment in which the application was to be deployed.

On the other hand, if a good component model is available to form the basis of the applications, both in terms of providing infrastructure as well as providing a programming model that defines a consistent component interface, the results of merging the applications may be substantially different. The

The Enterprise JavaBean component model allows components from multiple vendors to work together seamlessly.

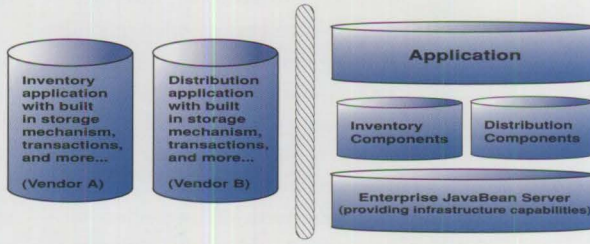


Figure 1. Traditional Development vs. Component Development

Enterprise JavaBean environment allows the vendors to focus on the business content rather than on the storage and transaction model. The result is a layered approach to building applications. On the bottom layer, a server provides a common way to install and run components. Next, the component layer (or layers) forms the heart of the application through the assembly of server-side business components. Finally, a client is assembled that uses various components from the business component layer.

In the end, because of enforcement of a common programming model, the components built for inventory look and behave very similarly to components built for distribution. Applications are able to pick and choose among the components to provide a specific value.

This model is proven to yield positive results, even with the Enterprise JavaBean specification being relatively young. The model provided by Enterprise JavaBeans is similar to that of the IBM SanFrancisco Business Component Framework. Customers using the SanFrancisco* Business Component Framework typically see a substantial increase in productivity when building the server-side business logic that forms the heart of their applications. Further, components built by IBM and supplied in the framework are being heavily reused by customers building applications with SanFrancisco.

For example as shown in ***Figure 2***, companies are often able to use many SanFrancisco "Common Business Object" (CBO) components (such as Currency, Exchange Rates, Company) and add their own custom built components to provide the function and capa-

CONTINUED ON PAGE 22

CONTINUED FROM PAGE 21

bilities that their applications need.

The current release of the IBM SanFrancisco Business Component Framework (Version 1, Release 4) is based on a proprietary foundation that is being replaced with an Enterprise JavaBean foundation.[2]

Using the Enterprise JavaBean specification for the basis of business components fulfills the need to be able to move the components to various operating environments quickly. In other words, Enterprise JavaBeans written to the specification from Sun should be able to move between Enterprise JavaBean servers quickly and effortlessly regardless of the operating system that the server is provided on, or the company that provides the Enterprise JavaBean server. This is the advantage of using the platform independent nature of the Java language.

Providing business content

A considerable amount of work was done – and is still being done – by IBM to evaluate business application requirements and build framework components that fulfill those requirements. The result of this work is the IBM SanFrancisco Business Component Framework. A primary goal of the EJB Business Component work is to maintain as much of the analysis and requirements from the IBM SanFrancisco Project as possible, but provide that function in a slightly different form. This form is a component independent of the context of the framework.

Both the Enterprise JavaBean (EJB) Business Components and the SanFrancisco Common Business Objects (CBOs) come from the same domain analysis. The content they provide is conceptually similar. The differences are in the amount of reusable content that is provided, the potential ability to scale to large applications, the amount of 'process' that automatically occurs between components, and the amount of technology that is built into the components themselves. In order to make this more concrete, look at the Currency category (a category is a group of closely related classes). The Currency category provides the support for:

Currencies: This allows you to define different currencies and their attributes. For example, defining US dollars having an identifier of USD, two decimal places, with the fractional part called cents.

Currency Values: This is a pairing of a currency with a decimal. For example, 31.56 USD. This is needed so that we don't accidentally add 10.00 to 31.56 only to find out later that one was Canadian dollars and the other was US dollars.

Transaction Values: This is a combination of two Currency Values. This supports management of transactions that occur in a currency foreign to your company. In particular, if your company is in Canada and is managed in Canadian dollars and you do business with a US company, you will bill them in US dollars, but want to track the equivalent Canadian dollars. For example, if the US company is billed 100 USD, then the Transaction Value would contain 100 US dollars and 150 Canadian dollars. Both are needed because the exchange rate between the two currencies fluctuates, and the Transaction Value captures the relationship between the two values at a particular point in time.

Exchange rates: This allows the definition of time based exchange rates between two currencies. (In our example, 1 US dollar is equal to 1.5 Canadian dollars from January 1st to January 19th, 1999.) Different types of exchange rates can be defined. For example, an application can set up one exchange rate table for intercompany transfer of money (usually set periodically) and another exchange rate table for customer invoices.

Both Currency categories (the one contained in the IBM SanFrancisco Business Component Framework and the one contained in the EJB Business Components) provide this basic support. The EJB Business Component's Currency category provides this support on a generic EJB server in a manner that is independent of other categories. This provides good basic reusable content that can be used to develop many different applications.

The SanFrancisco Currency category provides advanced support. For example, early release of the EJB Business Component's Currency category provides only time-based exchange-rate support, which minimizes coupling between categories. But, especially in large companies, time-based exchange rates are not sufficient. Instead, the ability to define the exchange rates based on the company's fiscal calendar is needed. This support is provided in the SanFrancisco Common Business Objects. This does not mean that this cannot

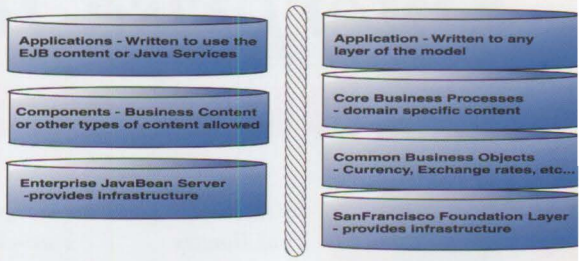


Figure 2. Enterprise JavaBean component model vs. SanFrancisco Business Component model

be added to the EJB Business Components – either later by IBM, or by an application developer based on the existing Currency category; it simply means it is available now and reusable as part of the SanFrancisco CBOs.

In both Currency categories, you can define a set of currencies that are valid. However, the EJB Business Components assume that you have one set defined that is applicable to your entire organizational hierarchy. On the other hand, the SanFrancisco CBOs provide support for using the organizational hierarchy to define what information is shared across the organizational hierarchy and what is applicable to only a piece of it. For example, you can define Canadian dollars, US Dollars and French francs as applicable for the entire organization and then add Italian lire and remove French francs for one particular company. Again, this capability can be built using the EJB Business Component mechanisms. It is not a capability that is available at the outset.

In addition to these extensions to the individual categories, SanFrancisco provides more reusable content at a higher level than the EJB Business Components and a more scalable technical infrastructure to build on that focuses on business capabilities rather than on generalized capabilities. The Core Business Processes (CBP) in the SanFrancisco Business Component Framework are the reusable core pieces of individual domains. Currently, SanFrancisco provides these for General Ledger, Accounts Receivable, Accounts Payable, Warehouse Management and Order Management. These CBPs provide a business flow through the business objects they provide. This is unlike the CBOs, which tend to provide building block capabilities for higher level business functions. For example, the Warehouse Management CBP builds on the SanFrancisco CBOs and provides support for managing Warehouses and their inventory.

SanFrancisco provides business scalability in a number of ways. One way is by scaling up to companies with complex organizational hierarchies. For example, SanFrancisco supports use of the organizational hierarchy to define application contexts as described in the currency example above. By specifying the organization you work for, you define what information you can see and what business policies you are using. The visibility of information is described above in terms of which currencies are valid. Another example of where SanFrancisco uses this mechanism is to define the chart of accounts available for a particular company. Again, this support can be added to the EJB Business Components; however, SanFrancisco has this support built in.

SanFrancisco's technical infrastructure also supports scaling to larger applications, not only because of the additional reusable content that it provides, but also in the business-application-driven extensions to the EJB specification that it provides. For example, SanFrancisco adds the ability to do general (not fixed) queries against a collection and have this general query pushed down to whatever database is being used for persistence. Pushing down a query can increase the performance of the query by several orders of magnitude.

Conclusion

Component models, when used effectively, can provide business content reuse without being tied explicitly to an operating system. By using these component models, two of which were described here, (the IBM SanFrancisco Business Component Framework and the Sun Enterprise JavaBean Specification) business components can be developed without explicit knowledge of infrastructure technology. The focus on business content can help in creating rapidly deployable applications and a substantial amount of reuse. Two forms of business content also were discussed here, content provided by the IBM SanFrancisco Business Component Framework, and future content under development by IBM.

The current alternatives of the Enterprise JavaBean Business Components and SanFrancisco allow you to decide what is appropriate for your particular needs. The Enterprise JavaBean Business Components can be used for early prototyping and education prior to working with the broader SanFrancisco product or used to build applications that do not need the advanced support provided by SanFrancisco. Either way, you can increase the speed with which you develop applications by reusing existing component technology, and using the programming models to build your own reusable components.

For additional information on SanFrancisco see Monday, Dangler, and Carey, *The SanFrancisco Framework: An Introduction*, Addison-Wesley-Longman, 1999.

[1] *This is a simplistic view of the process. For more information, see the Enterprise JavaBean specification available from Sun Microsystems at www.javasoft.com.*

[2] *More details on this transition are available at the IBM SanFrancisco Web site at www.software.ibm.com/ad/sanfrancisco/*

James Carey *is a Senior Software Engineer at IBM, where he has been involved with SanFrancisco Framework since its inception. He is currently the architect responsible for SanFrancisco's Common Business Objects and Core Business Processes. Jim is a coauthor of the book SanFrancisco Frameworks: An Introduction and is also working on a book describing the business patterns involved in the construction of the SanFrancisco Framework.*

Paul Monday *is an Advisory Software Engineer at IBM, where he is leading a team devoted to creating Enterprise JavaBean based business content based on SanFrancisco Business Component Framework content. Previously, Paul was instrumental in defining and refining the SanFrancisco client model, including the SanFrancisco Bean model and client infrastructure. Paul is a coauthor of the book SanFrancisco Frameworks: An Introduction.*

Advanced Edition:

Deploying BEA WebLogic *Enterprise JavaBeans* in WebSphere Application Server

by Paul Fremantle

IBM WebSphere Application Server Advanced Edition v3.0 is the IBM entry-level production Java Application Server. It runs transactional business objects written to the Enterprise JavaBeans 1.0 standard, as well as Java Servlets and Java Server Pages. It is a major enhancement over the previous version, and IBM believes that it is a major step forward with leading performance, availability and manageability. IBM has put years of experience in writing Object Transaction Monitors into building a Java-based transaction-enabled application server.

IBM WebSphere Application Server Advanced Editor 3.0 includes the following features:

- Full support for Java Servlets 2.1, Java Server Pages 0.91 (and 1.0 preview), Enterprise JavaBeans 1.0 including Container Managed Persistence
- Clustering support allowing high availability and scalability
- High performance (~ 2X performance improvement over v2.02) including performance enhancements
- Support for multiple Java Virtual Machines (JVMs) per node to provide high performance on multi-way symmetric multiprocessors (SMPs)
- Remote-Method-Invocation and Internet Inter-ORB Protocol (RMI-IIOP) support based on the Sun and Object Management Group (OMG) specifications
- Distributed transactions using two-phase commits and recoverable logging
- Support for Container Managed Entity Beans on Oracle and DB2 Universal Database (UDB)
- Advanced database optimizations (such as PREPARE statement caching)
- Optimized performance in-process servlet-to-EJB and EJB-to-EJB calls
- Support for complex persistence mapping when used with IBM VisualAge for Java 3.0 Enterprise Edition, including inheritance and associations
- Remote Systems Management interface giving a single system image of a cluster
- Distributed debugging interface and client
- Security using CORBA Secure Association Services

A number of large corporations are currently evaluating the Enterprise JavaBeans (EJB) specification as a potential standard for building new applications, for the Internet or intranet. Many of these customers have asked IBM how to port applications onto WebSphere from the BEA WebLogic server. This article is a guide for developers who wish to port EJB components – Enterprise JavaBeans – from WebLogic 4.0x to IBM WebSphere Application Server 3.0. The example that corresponds to this article is available at www.developer.ibm.com/devcon/mad.htm.

In theory, Enterprise JavaBeans are completely binary-compatible with any servers. However, because Enterprise JavaBeans are still a new technology, there are some minor differences between implementations. The EJB 1.1 specification provides for much "tighter" compliance, and should provide better interoperability. However, most implementations of EJB Servers are still running at the EJB 1.0 specification.

In general, there are four aspects of an EJB that may differ between implementations:

- The actual bean code, which is mainly well defined in the EJB 1.0 specification, should be compatible across implementations, with some minor exceptions.
- The EJB-JAR file, which also is defined in the standard, is designed to be a "binary" standard that EJB-JARs, developed to the EJB specification, are meant to be deployable on any EJB platform. The EJB-JAR file consists of the code, a manifest and the serialized deployment descriptor .ser file.
- The client code, which is mainly defined, should be compatible between implementations, with some minor exceptions.
- The deployment process, which is not defined in the EJB specification, differs between EJB implementations. The deployment process takes EJB code, packages it as an EJB jar file, creates support code (for example stubs and ties) so that it can be hosted in a particular EJB container, and, finally, installs it into the server. When the JAR file has been deployed, it has non-standard code in it. Different vendors all have slightly different views on exactly what the "deployment" process is. The above description presents the widest possible interpretation.

The aim of this guide is to explain what the differences in implementation are between WebSphere and WebLogic, and where these two systems differ from the published EJB 1.0 specification. It begins by describing the differences in the code and the deployment process in each system. Finally, an example of taking a WebLogic EJB sample and getting it running on WebSphere Advanced is presented.

If you use VisualAge for Java Enterprise Edition 3.0, you can take advantage of IBM enhancements to the EJB programming model, as well as a powerful persistence framework.

Code differences

The differences in the deployment process have been covered. However, there are also differences in the code required by WebLogic and WebSphere. Neither system is 100 percent EJB 1.0 compatible. The summary is:

WebLogic

- Uses standard Java casts (Object) rather than javax.rmi.PortableRemoteObject.narrow()

- Uses isModified() to specify which methods are read-only

WebSphere

- Requires the primary key to define hashCode() and equals(), which are part of the EJB1.1 specification.

Details

- WebLogic client code contains statements such as:

```
AccountHome home = (AccountHome) ctx.lookup(HOMENAME);
```

This is non-standard. The EJB Specification specifies that the RMI-IIOP syntax must be used for EJB access, although the EJB specification doesn't specify which wire protocol is used. The code should look like:

```
AccountHome home = (AccountHome)
    javax.rmi.PortableRemoteObject.narrow(
                                ctx.lookup(HOMENAME),
AccountHome.class);
```

In general, when casting a remote object to a derived subclass, you must use narrow().

- The isModified() code in WebLogic Enterprise JavaBeans is non-standard. WebSphere uses another approach, marking particular methods as read-only (for example, C++ const methods). The WebLogic code will be ignored by WebSphere.
- WebSphere requires primary keys of Entity Beans to explicitly define the hashCode and equals functions. This is required in all Enterprise JavaBeans in the EJB1.1 specification. The equals function must check if two keys are equal based on the properties and the hashCode must return equal values if the properties of two keys are equal. To continue the process described in this paper, add these functions to all Enterprise JavaBeans now.

If you use the VisualAge for Java EJB support these methods are created for you automatically. Otherwise, you must code them yourself.

Note: Enterprise JavaBeans without these methods will "install" and run with a single client, but may have transactional errors with multiple clients. Do not be tempted to leave them out.

An example of adding these methods to an existing WebLogic EJB is given in the next section.

Gaps in the EJB Specification

There are some areas where the EJB 1.0 specification partially defines the specification, leaving some areas undefined. The two major areas left undefined are:

- How to define the user's identity so that it can be passed to the server.
- How to define "finder" methods, so that Container Managed Persistence (CMP) Entity Beans can push down queries onto the database.

The WebLogic implementation of the security uses the JNDI specification to define a user's identity. This can be run on WebSphere without changes, although WebSphere ignores it. WebSphere uses a CORBA model for security. (Look at the WebSphere Documentation Center for further information.)

WebSphere Advanced uses a concept called a "FinderHelper" to define the finder logic. The following finder logic is required for each finder method (other than the findByPrimaryKey method) contained in the home interface of an entity bean with CMP:

- The logic must be defined in a public interface named NameBeanFinderHelper, where Name is the name of the enterprise bean (for example, AccountBeanFinderHelper).
- The logic must be contained in a String constant named findMethodNameQueryString, where findMethodName is the name of the finder method. The String constant can contain zero or more question marks (?) that are replaced from left to right with the value of the finder method's arguments when that method is invoked.

If you define the findLargeAccounts method shown below, you also must create the AccountBeanFinderHelper interface shown below. Note that to create the FinderHelper, you need to understand the mapping from the CMP bean to the database.

The AccountBeanFinderHelper interface for the EJB server

There are some areas where the EJB1.1 specification defines the states of Enterprise JavaBeans much more clearly, and there may be different interpretations among EJB1.0 vendors. One such example, concerns which methods are allowed on the Context objects, depending on the state of the bean.

```
...
public interface AccountBeanFinderHelper{
  String findLargeAccountsQueryString =
    "select * from ejb.accountbeantbl where balance > ?";
}
```

Implementation-specific changes

There are some changes that are implementation specific. The JNDI context factory name and the URL to access the naming service are specific to each server. In general, these should be supplied in property files or in the environment, instead of being hardcoded.

Recap of deployment with WebLogic

Deployment in WebLogic as shown in ***Figure 1***, goes along the following lines:

a) Develop the EJB code.

b) Create a textual deployment descriptor.

c) Either use the DDCreator tool to create a .ser file, and use jar to package it as an EJB-JAR file, or use the EJB deployment wizard to create and edit the serialized deployment descriptor and EJB-JAR.

d) Create the wrapper classes, either in the deployment wizard or using the ejbc compiler.

e) Install by editing weblogic.properties.

This is a very straightforward process. In theory, it would be nice to take the EJB-JAR file created in step c and deploy it in WebSphere. Unfortunately, the JAR file, and .ser file created aren't standard EJBs. The serialized DD is meant to be a serialized object of class javax.ejb.EntityDescriptor or javax.ejb.SessionDescriptor. The WebLogic tools create a serialized object of class weblogic.ejb.utils.WLEntityDescriptor or the session equivalent, because WebLogic has features that the EJB specification doesn't define. It uses the DD as a way of defining these, creating its own DD format on the way. To use a WebLogic EJB elsewhere, you must go back to step a and use the original EJB code.

Let's now review the deployment process under WebSphere.

Deploying Enterprise JavaBeans in WebSphere

Deploying Enterprise JavaBeans for WebSphere Advanced 3.0 can be done in two different ways: either using the WebSphere native tools (jetace), or using VisualAge for Java Enterprise edition. Using VAJava with WebSphere adds a number of features and some extensions over the EJB specification, including:

- Mapping Container Managed Entity Beans (CMP-EJBs) to existing databases (including

CONTINUED ON PAGE 26

CONTINUED FROM PAGE 25

cross-table joins)

- Creating relational links (associations) between CMP-EJBs
- Inheritance of Enterprise JavaBeans including CMP-EJBs
- Marking const methods as read-only to improve performance

While these extensions can be used, VA Java always offers the option of generating pure standard EJB-JAR files. Deploying Enterprise JavaBeans inside VAJava is described in the VAJava documentation, and deploying inside WebSphere is described in the *Writing Enterprise Beans in WebSphere* book.

Deploying using WebSphere tools

The WebSphere Advanced Edition v3.0 deployment tool is called JetAce. JetAce can do three things:

1. Create a new standard EJB-JAR from a set of EJB classes.
2. Edit an existing deployment descriptor stored in an EJB-JAR.
3. Import and export deployment descriptors in XML format.

Note: The XML format used in JetAce predates the EJB 1.1. XML deployment descriptor format. A later version of WebSphere will support the EJB 1.1. XML format (see ***Figure 2***).

The XML format has the same purpose as the textual deployment descriptor in WebLogic. WebSphere/JetAce also allows the developer to create a deployment descriptor visually. If Java IDE allows exporting of JAR files, or the developer is happy using the jar utility that comes with Java, an initial JAR file can be created that includes all the EJB class files and any support classes that are required (such as exception classes). The jar file can contain multiple Enterprise JavaBeans. To see an example of using JetAce, refer to the example later in this paper.

Deployment within VA Java Enterprise 3

VA Java Enterprise edition (Enterprise Update) also includes tools to edit a Bean's deployment descriptor as well to export a standard EJB-JAR file. Those details are not covered here. VA Java can be used to specifically target WebSphere Application Server, and in that case there is some extra functionality available (see ***Figure 3***):

- Meet-in-the-middle (and bottom-up) object to relational mapping – using WebSphere Application Server alone, CMP Entity Beans are mapped into new tables that WebSphere creates to store the CMP data. The mapping is a straightforward, top-down mapping – one bean to one table. If you use VA Java, you can do complex mappings into existing databases. These mappings include joins, complex mappings from objects into multiple columns, and custom datatype converters.
- Inheritance of Enterprise JavaBeans, including mappings to existing tables – the EJB model does not define how one EJB inherits from another. VA Java enhances WebSphere with an inheritance schema that supports building proper object models from Enterprise JavaBeans.
- Associations between beans – where CMP Entity beans have relationships between each other, there needs to be support to allow navigation and relational integrity. VA Java supports building EJB models that include this association.
- Read-only/const methods. This allows the developer to mark certain methods as read-only, allowing WebSphere to make major performance improvements (e.g. not writing the same data back to the database). This provides equivalent functionality to the isModified extension that WebLogic supports, except that it doesn't involve code changes.
- EJB Access Beans – VA Java can automatically generate JavaBean client objects that provide simplified access to Enterprise JavaBeans for applets, applications and Java Server Pages.

While the WebLogic approach involves an extended deployment descriptor, the VAJava / WebSphere link for enhanced functionality is based on smart extensions to the code generation step built into VAJava. If you want to take advantage of these extensions, you need to do the code generation (equivalent to ejbc) inside VAJava. This is the "Generate Deployed Code" step. Then export the "EJS-JAR" file. This is an enhanced EJB-JAR file

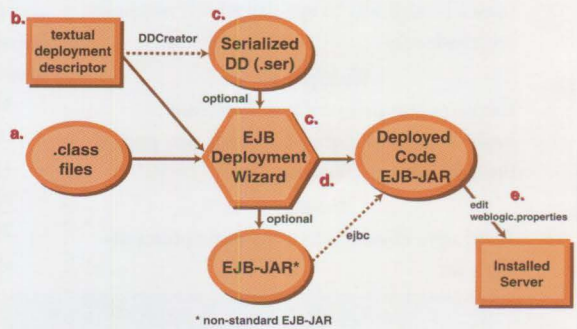


Figure 1. Deployment with WebLogic tools

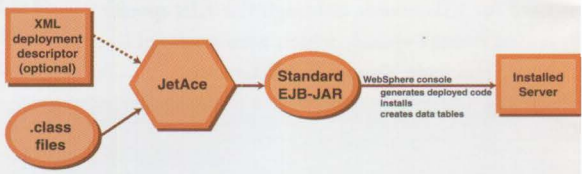


Figure 2. Deployment with WebSphere tools

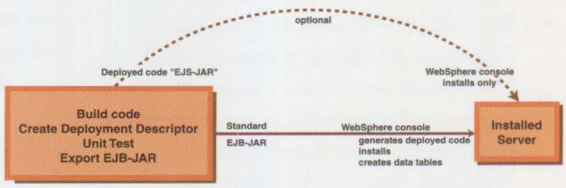


Figure 3. Deployment with VisualAge Java

that includes the non-standard enhancements, and can be directly installed into WebSphere. WebSphere notices that the EJS-JAR file has extensions, and skips the code generation step. ***Note:*** A standard EJB-JAR file can be exported from VA Java at any stage.

Deployment summary

To summarize, because WebLogic does not include a facility to generate a standard EJB-JAR file, you must always start the WebSphere deployment from the existing code (.java or .class files). Then visually generate the deployment descriptor, and export the file as an EJB-JAR file, that can be installed into WebSphere using the Admin Console. If you use VisualAge for Java Enterprise Edition 3.0, you can take advantage of IBM enhancements to the EJB programming model, as well as a powerful persistence framework.

To see this complete article and its example, go to the Developer Connection Web site at www.developer.ibm.com/devcon/.

Paul Fremantle is a technical specialist and architect working with IBM WebSphere, VisualAge Java, Enterprise JavaBeans and XML to build leading-edge Web sites. Paul has been developing Web applications since 1995. Before that he developed groupware and forecasting software. He has a BA in Mathematics and Philosophy and a MSc in Computation from Oxford University.

IBM jStart Program for XML

NOW THAT XML IS MATURING, MANY FIRMS ARE TAKING THE NEXT STEP AND CAPITALIZING ON THE BENEFITS OF XML TO THEIR BUSINESS. ARE YOU INVESTIGATING THE USE OF XML IN YOUR MISSION-CRITICAL APPLICATIONS? COULD YOUR INDUSTRY OR BUSINESS BENEFIT FROM A COMMON BUSINESS LANGUAGE THAT WOULD ALLOW YOUR ENTERPRISE SYSTEMS TO INTEROPERATE? ARE YOU LOOKING TO FRONT-END YOUR LEGACY ENTERPRISE SYSTEMS WITH DYNAMIC WEB APPLICATIONS?

IF YOU HAVE ANSWERED YES TO ANY OF THESE QUESTIONS, THEN IBM'S JSTART PROGRAM MAY BE ABLE TO HELP.

IBM: Taking the mystery out of new technologies

TO HELP IBM CUSTOMERS GAIN THE CRITICAL EXPERIENCE THEY NEED USING EMERGING AND ESTABLISHED TECHNOLOGIES, IBM HAS CREATED THE "JSTART" (JUMP-START) PROGRAM. TECHNOLOGIES SUCH AS XML, JAVA, PERVASIVE COMPUTING, AND ENTERPRISE JAVABEANS HAVE MATURED AND ARE NOW BEING USED BY HUNDREDS OF FORTUNE 1000 COMPANIES TO PERFORM MISSION-CRITICAL TRANSACTIONS. THE JSTART PROGRAM HELPS IBM CUSTOMERS GET STARTED ON AN APPLICATION THROUGH A UNIQUE PARTNERSHIP THAT BRINGS THE RIGHT COMBINATION OF TOOLS, SKILLS, AND RESOURCES TOGETHER FOR THE RAPID DEPLOYMENT OF PRODUCTION APPLICATIONS.

THE IBM JSTART PROGRAM: USING A PROVEN, 5-STEP ENGAGEMENT MODEL, THE JSTART TEAM CAN WORK WITH YOU TO IDENTIFY AND QUALIFY A PROJECT. KEY CRITERIA TO YOUR PARTICIPATION IN THIS PROGRAM IS THAT YOU ARE WILLING TO BE A REFERENCE ACCOUNT FOR THE JSTART PROGRAM AND IBM. THE PROVEN 5-STEP ENGAGEMENT MODEL HAS BEEN USED BY COMPANIES SUCH AS: SABRE GROUP, FIRST UNION BANK, JB HUNT, SCOTTISH EQUITABLE, GENERAL DYNAMICS' ELECTRIC BOAT DIVISION, QUALCOMM, TOKUSHIMA SHINBUN, UNIVERSITY OF MINNESOTA, CREDIT AGRICOLE, AND MANY MORE.

THE JSTART TARGET AUDIENCE: AS GEOFFREY A. MOORE DESCRIBES IN HIS BOOK, *CROSSING THE CHASM*, THE TECHNOLOGY ADOPTION LIFE CYCLE MODELS THE MARKET PENETRATION OF ANY NEW TECHNOLOGY IN TERMS OF A PROGRESSION IN THE TYPES OF CONSUMERS THE TECHNOLOGY ATTRACTS THROUGHOUT ITS USEFUL LIFE.

THE TWO KEY GROUPS IN THE LIFE CYCLE MODEL ARE THE EARLY ADOPTERS AND THE EARLY MAJORITY. FOR ANY EMERGING TECHNOLOGY TO EXPERIENCE A COMPLETE TRIP THROUGH THE LIFE CYCLE IT MUST SUCCESSFULLY TRANSITION FROM THE EARLY ADOPTERS TO THE EARLY MAJORITY.

THE OBJECTIVE OF THE JSTART PROGRAM IS TO HELP MAKE THAT TRANSITION BY FOCUSING ON THE AUDIENCE COMPRISED OF THE EARLY ADOPTERS AND THE EARLY MAJORITY.

The benefits of jStart:

- IMPROVED SKILLS THROUGH MENTORING. RAPID SKILLS DEVELOPMENT AND KNOWLEDGE TRANSFER FROM IBM OO DESIGN AND ARCHITECTURE (IGS AND PRODUCT SERVICE SPECIALISTS) EXPERTS TO YOUR STAFF. PARTNERING THROUGH EDUCATION, IBM MENTORS YOUR TEAM.
- MINIMIZED RISK. THE PROVEN 5-STEP ENGAGEMENT PROCESS HELPS LOWER THE RISK OF DEVELOPMENT FOR YOUR PROJECT. THIS PROCESS HAS BEEN EFFECTIVELY AND EFFICIENTLY USED BY COUNTLESS IBM CUSTOMERS AND IS A MEANS TOWARDS ACHIEVING A SUCCESSFUL APPLICATION.
- BUSINESS FOCUSED. THE JSTART PROGRAM ALLOWS YOU TO FOCUS ON MEETING BUSINESS OBJECTIVES AND THE DELIVERY OF THE APPLICATION INSTEAD OF DEVELOPMENT TOOL FEATURE/FUNCTION COMPARISONS.
- START SMALL AND GROW FAST. THE JSTART PROGRAM HAS ALLOWED IBM CUSTOMERS TO "GET A STEP AHEAD" WITH E-BUSINESS APPLICATIONS. TAKING ADVANTAGE OF NEW TECHNOLOGIES, INCLUDING JAVA, XML AND EJB, CUSTOMERS CAN REALIZE THE BENEFITS OF THIS MULTI-PLATFORM DEVELOPMENT ENVIRONMENT FOR APPLICATIONS SUCH AS EXTRANET-BASED SUPPLY CHAINS, INTERNET CUSTOMER CARE AND SELF SERVICE, AS WELL AS FINANCIAL AND ACCOUNTING.

CONTACT JSTART@US.IBM.COM FOR MORE INFORMATION.

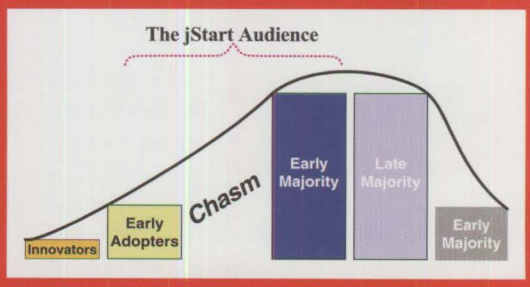


Figure 1. Technology adoption life cycle

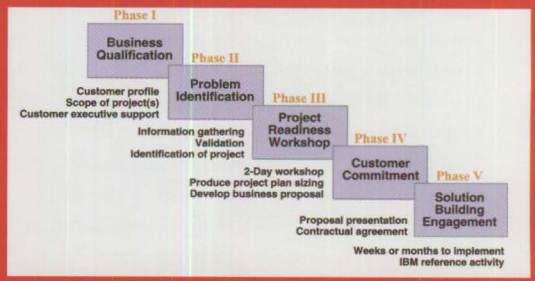


Figure 2. jStart engagement model

The VisualAge for Java repository

by Alfredo Gutierrez

The VisualAge for Java repository is both a source control mechanism and a storage facility for code. When you start VisualAge for Java, your interface or the "workspace" is connected to the repository so that you can create or update program elements through the workspace and your changes are stored in the repository. Understanding the repository is probably the biggest leap that you need to make when first becoming familiar with VisualAge for Java. Program elements are stored only in the repository, not in the file system. You can access the repository through the workspace. You must add program elements to the workspace before you can modify them. Every program element that is added to the workspace is stored in the repository and VisualAge for Java builds pointers from the workspace to the source code in the repository. When you remove a program element from the workspace, it still remains in the repository but can only be browsed.

The following is a list of differences between the workspace and the repository:

- The workspace contains bytecode. The repository contains source code and Visual Composition Editor information.
- Only one edition of any program element is present in the workspace at any time. The workspace should contain only the program elements that are currently being worked on. The repository contains every edition of every program element that has ever been developed (until the repository is compacted).
- Use the Workbench window to view, manipulate, create, modify, and manage program elements that are in the workspace. The Repository Explorer window is used to view program elements that are in the repository, add them to the workspace, and purge them from the repository.
- Changes to the workspace are not saved until you select **Save Workspace** from the **File** menu, or until you exit from VisualAge. Changes to the source repository are saved immediately, every time you save changes to a method, class, or interface.

Because the repository maintains all editions of all program elements, when program elements are removed from the workspace, they are not removed from the repository. The result is that the repository grows larger and larger over time. (Later, this article explains how program elements that are no longer required can be periodically purged and the repository compacted to reduce its size.) As both the source control mechanism and storage facility, the importance of the VisualAge for Java repository cannot be overstated. After providing some necessary background information about the workspace and program element editions, this article explains the characteristics, features, and benefits of the repository and describes techniques for managing it.

Workspace

Before proceeding, let's review some basic features of the workspace and describe how it relates to the repository. The VisualAge for Java workspace is contained in a file (ide.icx)

Understanding the repository is probably the biggest leap that you need to make when first becoming familiar with VisualAge for Java.

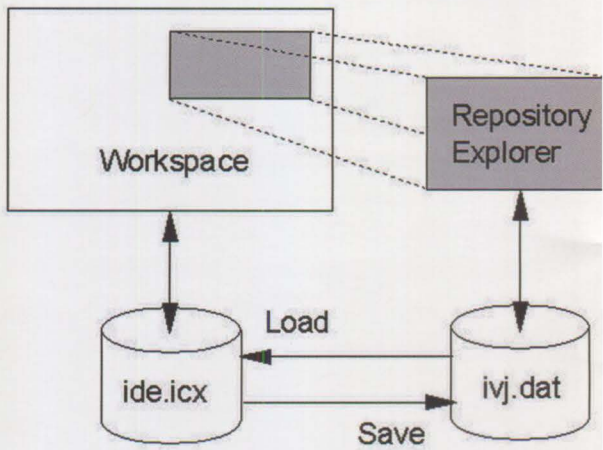


Figure 1. The Repository Explorer

that stores all the program elements (projects, packages, classes, methods, interfaces) currently being worked with. In addition, the workspace also holds the user's preferred optional settings. Source code is stored in a larger database called the repository. The program elements in the repository are stored in a defined state – an edition. You can move program elements from the repository to the workspace if they need to be used or modified. When you start VisualAge, Java code is loaded from the repository into the workspace. As you work, code is automatically saved from the workspace to the Repository as shown in ***Figure 1***.

When you close the last opened window of VisualAge for Java, the IDE prompts you to confirm that the workspace should be saved. Canceling keeps VisualAge for Java from closing. You can save the configuration of the workspace at anytime by selecting **File –> Save Workspace**. You also can exit from VisualAge for Java by selecting **File –> Exit VisualAge**.

Editions

VisualAge for Java maintains several editions of a program element. As you save program elements, either in the workspace or in the repository, VisualAge for Java keeps track of the code. The code you are working on is saved in an edition. An edition is the state, at a particular point in time, or "snapshot" of a particular program element. Whenever a project, package, or class is a work-in-process, it

is actually a specific edition of a program element that is present in the workspace and being worked with. Only one edition of a program element can be present in the workspace at any given time. It is possible to see which editions are in the workspace by clicking the Show Edition Names button.

An open edition is a work-in-progress. Before it becomes available for changes, an existing project, package, or class must be available as an open edition. Open editions appear in VisualAge for Java windows with a time-stamp, in parentheses, showing when they were created.

Versioned editions are editions that cannot be changed. However, versioning does not prevent someone from ever changing a program. To make changes, you must create a new open edition of the program element. You can revert to an earlier version of a program edition by replacing the edition in the workspace with a different edition from the repository. You might want to create a versioned edition for any of the following reasons:

- To keep a copy of a program element at some meaningful point so it can be returned to at a later date. In the case of packages and projects, versioning freezes a specific configuration of the contained program elements, which must also be versioned.
- To make changed classes available to other team members who are browsing the repository.
- To release a class into its package. Classes must be versioned before they can be released.

Versioned editions appear in VisualAge for Java panels with version names, rather than as open editions that appear with time-stamps. When you version an open edition of a program element, the name can be assigned by VisualAge for Java automatically or you can choose a name. It is possible to see all of the available editions of a Project, Package, or program element by using the Repository Explorer.

Repository Explorer

Use the Repository Explorer to view the contents of your repository file. You can open the Repository Explorer, shown in ***Figure 2***, by selecting **Window –> Repository Explorer**.

A breakdown of the editions, packages, and types within the packages is shown. There is a Projects tab and a Packages tab so you can switch between the views. The Projects view shows all projects. To show the editions in the repository, click the project and the editions listed to the right of the project. When you select an edition, the packages in the selected edition are displayed. If you select the package, the types (interfaces or classes) in the package are listed. It is possible to see all the editions of a specific type by selecting Open from the Types menu and clicking the Editions tab. It is also possible to compare two editions in the Projects or Packages view:

1. Select two editions.
2. Select **Editions –> Compare** from the menu bar.

As you develop projects, there will be times when you no longer need the program elements that are in the repository. For example, you might develop classes that are used for testing purposes and aren't needed after testing, or you might have projects that reach a reliable state and no longer need to keep prior editions of it. Purging unwanted program elements and compacting the repository not only reduces the size of the ivj.dat file but helps to increase the performance of VisualAge for Java.

Purging

Purging a program element marks that program element for later deletion from the repository. The purge function is on the pop-up menu in the Repository Explorer. (See ***Figure 3***.)

Because the program element is only marked for deletion but not actually removed, it is possible to retrieve a purged element. You can continue to restore purged elements until you compact the repository. The size of the repository file is not reduced simply by purging an element. It is necessary to compact the repository to actually delete the elements that have been purged or marked for deletion.

Compacting

Compacting the repository reduces its size by eliminating the following program elements:

- All open editions.
- All purged editions.
- Versioned editions of classes contained in open editions of packages.

A compacted repository that contains only versioned editions will be significantly smaller than the original. The procedure for compacting a repository is different depending on whether you use a local repository or a shared team repository. If you use VisualAge for Java, Enterprise Edition, you may be using a team repository. Either way, as a precaution, back up the repository (ivj.dat) before compacting it. To compact the repository, you must work from the Repository Explorer.

1. As a precaution, back up the repository (ivj.dat file) before compacting it.
2. Version any open editions of projects, packages, and classes that you want to keep.
3. Purge any projects and packages that you want to discard.

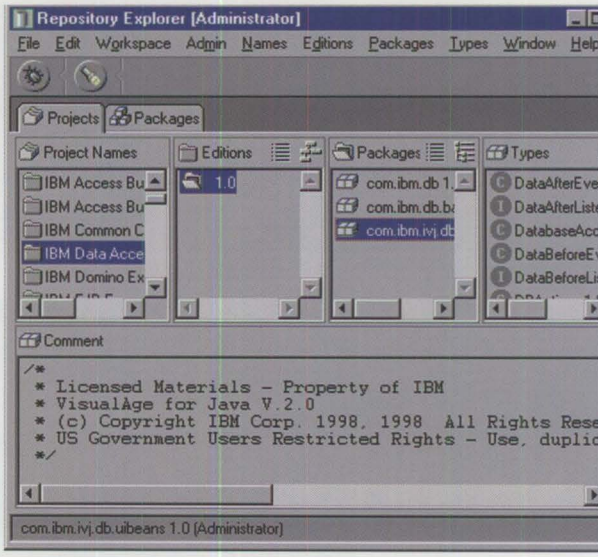


Figure 2. Repository Explorer (Administrator window)

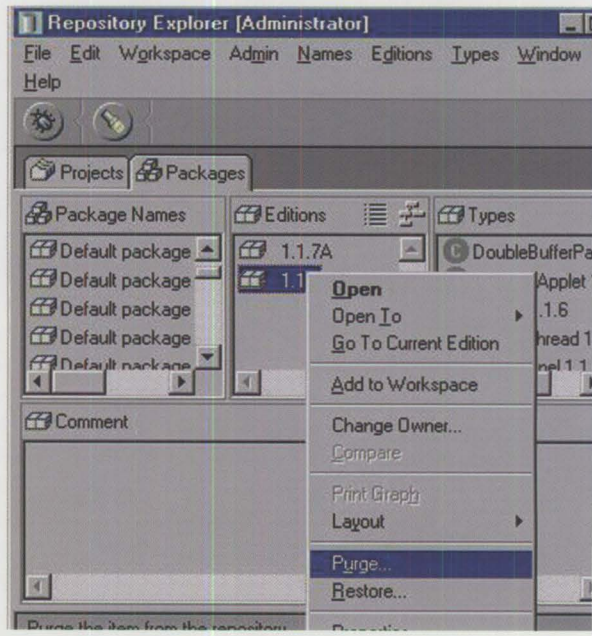


Figure 3. Purge selection on the pop-up menu

CONTINUED ON PAGE 30

CONTINUED FROM PAGE 29

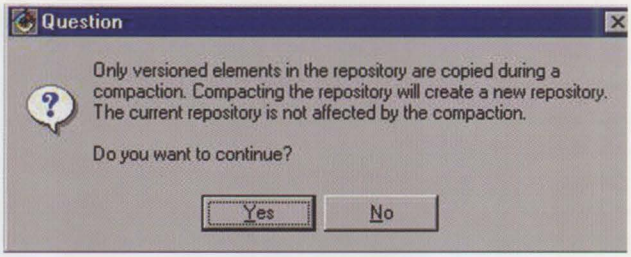


Figure 4. Compact panel question

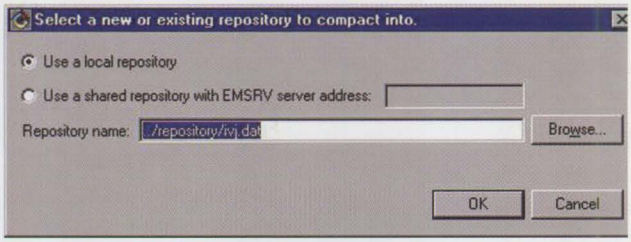


Figure 5. Select a repository

4. Open the Repository Explorer window.
5. From the Admin menu, select **Compact Repository**. The question panel shown in ***Figure 4*** appears.

Click Yes to proceed. If the Enterprise Edition is being used, another dialog, shown in ***Figure 5***, appears. Press OK to compact locally, or provide a new filename to create another local repository file, or choose to save the compacted repository to a shared repository on a server (you must have administrator authority).

Backing up the repository

The repository is where all of your source code is stored. The repository file that is provided with VisualAge for Java is called ivj.dat. You should back up this file regularly.

Begin to back up the repository by shutting down VisualAge in order to disconnect the workspace from the repository. When VisualAge has been shut down, use operating system commands to back up \IBMVJava\Ide\repository\ivj.dat. To restore the repository, exit the IDE and copy your backup repository file to ivj.dat.

Resource files, such as audio clips or image files, are not stored in the repository. VisualAge for Java stores these in the file system. If the repository contains applications that use resource files, back up those files at the same time you back up the repository.

Summary

The VisualAge for Java repository is fundamentally different from the storage systems found in other Integrated Development Environments. Understanding the repository is the biggest leap you need to make when switching to VisualAge for Java. Once you are familiar with how the repository works, however, you can quickly see its advantages. Every program element that is added to the workspace is stored in the repository. The repository maintains all editions of all program elements. When program elements are removed from the workspace, they are not removed from the repository. As both the source control mechanism and storage facility, the importance of the VisualAge for Java repository cannot be overstated.

Alfredo Gutierrez *is a Network Computing Consultant with IBM's Solution Developer Technical Support organization. Fred joined IBM in 1987 in Atlanta while attending Georgia State University. Upon graduation, he was transferred to Florida where he developed OS/2 and TCP/IP education at the International Technical Support Center in Boca Raton. In 1990 Fred was transferred to IBM's Austin site where for the last two years he has been involved in consulting and the development and delivery of Application Framework education to Solution Developers. In addition to supporting developers as a Solutions Architect, Fred also teaches lab-intensive e-business courses, including VisualAge for Java, at all the Solution Partnership Centers worldwide.*

DIRECTORY

IBM Developer Connection Technical Magazine

Order numbers in the following countries:

Austria	0660 8705	Mexico (D.F.)	387-5900
Canada	800-561-5293	Mexico (Interior)	01-800-006-3900
Germany	0 130 812177	United States	800-6DEVCON(633-8266)

Latin and South America order numbers:

Argentina	0-800-44-426 92Interno 2987	El Salvador	02-98 5011
Bolivia	02-35 1840	Guatemala	02-31 5859
Brazil	0800-111426 r.1351	Honduras	32-2319
Chile	800 218 218 anexo 6970	Panama	02-639 977
Colombia (Nacional)	9800-17555	Paraguay	444-094
Columbia (Bogota)	616-7555	Peru	349-0040
Costa Rica	223-6222	Uruguay	0-800-A-IBM (0-800-2-426)
Dominican Rep.	566-5161	Venezuela	800-1-5000
Ecuador	(02) 565-090		

Asia/Pacific order numbers:

The Developer Connection can be ordered in Asia/Pacific countries from IBM in Australia (61 is the country code) and Japan. Please ensure that you dial the international access code applicable to your country before the listed phone or fax number.

Australia:	Phone	+61 2-9354 7684
	Fax	+61 2-9354 7766
	E-mail	pwdap@au1.ibm.com
Japan:	Fax	+81 3-5200 6310
	E-mail	os2pid@jp.ibm.com

Europe and other countries not listed:

The Developer Connection can be ordered directly from IBM in Denmark (45 is the country code). Please ensure that you dial the international access code applicable to your country before dialing the appropriate phone or fax number. Operators speaking the following languages are available:

Danish	+45 48 101300	German	+45 48 101000
Dutch	+45 48 101400	Italian	+45 48 101600
English	+45 48 101500	Norwegian	+45 48 101250
French	+45 48 101200	Spanish	+45 48 101100
Finnish	+45 48 101650	Swedish	+45 48 101150
		Fax	+45 48 142207

Note: To subscribe to the Developer Connection in South Africa, contact Claudia Wissler, IBM South Africa, at 27-113029111, ext. 6960.

Electronic Support for the Developer Connection Release 2 Program

Electronic support is provided through the Internet and OS/2 BBS. Obtain technical support or use the forums to exchange messages, ideas, or comments with the Developer Connection team or other subscribers. Note: Refer to the specific product information for the operating system technical support methods.

Internet users may address their questions or comments to devcon@us.ibm.com.The DEVCON CFORUM is on the OS/2 BBS under TalkLink, which is a feature under the IBMLink Commercial Services. For TalkLink access, U.S. customers can call 1-800-426-5465; customers outside of the U.S. should contact their local IBM Marketing Representative. Note: Commercial members of PartnerWorld for Developers (formerly IBM Solution Developer Program) can access the DEVCON CFORUM on the Web from the PartnerWorld for Developers Web page (www.developer.ibm.com) without specifically signing up for IBMLINK.

PartnerWorld for Developers

The Developer Connection is one of a wide range of offerings that are available as part of PartnerWorld for Developers. In one single resource, PartnerWorld for Developers provides technical, business, marketing, and information services that can help you to reduce your development costs, accelerate your development efforts and showcase and sell your products. Members receive, at no charge, a wealth of information about IBM products and technology that can be searched on the PartnerWorld for Developers Web site 24 hours a day, seven days a week, at www.developer.ibm.com/

If you are not a member of PartnerWorld for Developers, why not join now – registration is free! It takes only a few minutes to register on the Internet, and in return, you'll receive a membership ID and password to access our services. Welcome aboard!

PartnerWorld for Developers Hotline:

United States/Canada	1-800-627-8363
Worldwide	1-770-835-9902
Worldwide Fax	1-770-835-9444

Global Solutions Directory:

The Global Solutions Directory is IBM's comprehensive, online source for over 30,000 partner solutions. The guide is a worldwide resource for applications, tools and services. Visit the Guide today at www.software.ibm.com/solutions/isv/

Staff

Publisher	Karen Foley
Managing Editor	Jean Swanson
Assistant Editors	Angela Ausman, Tracey Marcelo and Loree Eubank
Graphic Design	Stil Point Images, Michael Rainer

IBM

Get the e-business tools and information you need all year long.

The IBM Developer Connection was created to help you power your solutions – power them through development, power them to market, and make them powerful when they arrive. In a single resource, the Developer Connection merges the tools and strategies that you need to get up and running with the latest IBM technologies. And it provides core strengths you need to move solutions to new platforms, rapidly adopt breakthrough technologies and satisfy the challenging demands of your customers.

The Developer Connection is the single most comprehensive resource available to developers exploiting IBM technologies. Depending on your level of participation, your subscription to the Developer Connection provides:

- Fast, convenient, one-source access to more than 1,000 leading-edge development tools to help you build solutions on 14 platforms;
- The backing of IBM's e-business strategic initiative and ongoing support in such hot technologies and platforms as Java, XML, WebSphere, Linux and Windows NT;
- Web access to products, tools, technical tips and the very latest insights and information from IBM;
- CD sets giving you the flexibility to access the Developer Connection information in the way that's right for you; and
- *IBM Developer Connection Technical Magazine* – a quarterly collection of technical articles, news briefs and "heads up" previews of the latest technology and strategies from IBM.

Commercial members of PartnerWorld for Developers can view or download all content using their member id and password. Commercial members also can receive the IBM Developer Connection CDs for just the cost of manufacturing, shipping and handling.

e-business

Choose the membership level that works for you...

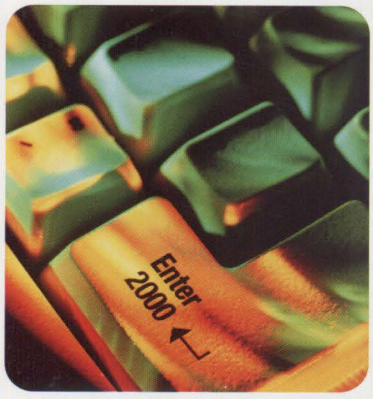
You can tailor your participation in the Developer Connection to match your interests and requirements. Your subscription entitles you to unlimited Web access for all content in your subscription level. A set of CDs is available for Member, Advanced and Premier levels.

Guest Level is available on the Web to any registered application developer. From our Web site at www.developer.ibm.com/devcon/, you can register, review and download sample source code, technical documents, hints and tips, utilities and Java and Internet tools. ***All free of charge.***

Member Level includes the Guest Level contents, while giving you the additional value and convenience of a CD collection, a Java-enabled browser, additional documents and non-IBM tools. At the Member Level, you'll also receive a subscription to the *Developer Connection Technical Magazine.*

Advanced Level builds on the Member Level by also providing compilers, toolkits for IBM operating systems and IBM e-business Servers. With an Advanced Level subscription, you have the tools to develop Java, Internet, e-business Servers or purely operating system applications.

Premier Level further extends the Advanced Level and adds system management tools and a comprehensive test environment for IBM Software Server applications.



Subscribe Today:

US	1-800-6DEVCON (633-8266)
Brazil	0800-111 426 r. 1351
Canada	1-800-561-5293
Argentina	0-800-44-426 92 Interno 2987
Asia/Pacific	+61 2-9354 7684
Venezuela	800-DE-IBM (800-33-426)
Europe	+45 4-810 1500
Japan	os2pid@jp.ibm.com
Mexico	387-5990 (D.F.)
Colombia	9800-17555 (Nacional)
Austria	0660 8705
Germany	0 130 812177

Phone numbers for customers in other countries are listed on page 31.

Printed on Recycled Paper
CT6T3NA